

LEONARDO LOPES RESENDE BARBOSA

PLATAFORMA DE STEWART:

**PROJETO DE UM MODELO SIMULADO E
CONSTRUÇÃO DE UM PROTÓTIPO SIMPLIFICADO**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

Uberlândia 2017

LEONARDO LOPES RESENDE BARBOSA

PLATAFORMA DE STEWART:

**PROJETO DE UM MODELO SIMULADO E CONSTRUÇÃO DE UM
PROTÓTIPO SIMPLIFICADO**

Trabalho de Conclusão de Curso,
apresentado como exigência parcial
para obtenção de grau de **BACHAREL
EM ENGENHARIA MECATRÔNICA**,
à Faculdade de Engenharia Mecânica
– FEMEC, da Universidade Federal de
Uberlândia.

Orientador: Prof. Dr. José Jean-Paul
Zanlucchi de Souza Tavares.

UBERLÂNDIA – MG
2017

LEONARDO LOPES RESENDE BARBOSA

PLATAFORMA DE STEWART:

PROJETO DE UM MODELO SIMULADO E CONSTRUÇÃO DE UM PROTÓTIPO SIMPLIFICADO

Trabalho de Conclusão de Curso,
apresentado como exigência parcial
para obtenção de grau de **BACHAREL
EM ENGENHARIA MECATRÔNICA**,
à Faculdade de Engenharia Mecânica
– FEMEC, da Universidade Federal de
Uberlândia.

Orientador: Prof. Dr. José Jean-Paul
Zanlucchi de Souza Tavares.

BANCA EXAMINADORA

Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares – UFU
Orientador

Prof. Msc. Alexandre Rodrigues de Sousa – UFU
Examinador

Prof. Dr. Rogério Sales Gonçalves – UFU
Examinador

UBERLÂNDIA – MG
2017

AGRADECIMENTOS

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica pela oportunidade de realizar este curso.

Aos colegas do Laboratório de Planejamento Automático de Manufatura (MAPL) pelo apoio e sugestões durante as apresentações prévias do trabalho.

Aos colegas da Equipe de Desenvolvimento em Robótica Móvel (EDROM) por fornecer parte do material para confecção do protótipo simplificado.

LOPES RESENDE BARBOSA, Leonardo. **Plataforma de Stewart**: Projeto de um modelo simulado e construção de um protótipo simplificado. 2017. 185p. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2017.

RESUMO

A Plataforma de Stewart é um mecanismo paralelo que apresenta vantagens com relação aos mecanismos seriais, mas possui grande complexidade na determinação de seu modelo cinemático direto. Este trabalho apresenta a análise de um modelo cinemático direto que, a partir das seis variáveis de entrada das posições dos atuadores, resulta em um sistema de 3 equação não lineares e 3 incógnitas, resolvido por método numérico. Além disso, realiza-se também a análise do modelo cinemático inverso, bem como a implementação de ambos os modelos cinemáticos em um protótipo simplificado utilizando o mecanismo de biela-manivela. O trabalho apresenta também uma análise de uma outra plataforma com atuadores lineares. Este outro modelo é simulado, e é concebido a partir das ferramentas SimMechanics® e Simulink® do MATLAB®, sendo controlado por PID. Também foi realizado o dimensionamento do curso e das capacidades de carga necessários aos atuadores hidráulicos para atender a requisitos especificados para o modelo.

Palavras-chave: Plataforma de Stewart. Mecanismo paralelo. Cadeia cinemática fechada. Simulação de mecanismos.

LOPES RESENDE BARBOSA, Leonardo. **Stewart Platform**: Project of a simulated model and construction of a simplified prototype. 2017. 185p. Course Conclusion Paper (Mechatronic Engineer Graduation) – Federal University of Uberlândia, Uberlândia, 2017.

ABSTRACT

The Stewart Platform is a parallel mechanism that presents advantages over serial mechanisms, but it has great complexity in determining its direct kinematic model. This work presents the analysis of a direct kinematic model that, from the six input variables of actuators position, results in a set of 3 non-linear equations system and 3 incognitos. In addition, the inverse kinematic model analysis is also performed. Both are implemented in a simplified prototype using crank-rod mechanism. The work also presents the analysis of another platform with linear actuators. This other model is simulated and is conceived from the SimMechanics® and Simulink® tools of MATLAB®, being controlled by a PID algorithm. The hydraulic actuators' dimensions and load capacities are also determined in order to meet pre-defined requirements.

Keywords: Stewart Platform. Closed kinematic chain. Parallel mechanism. Simulation of mechanisms.

SUMÁRIO

1 INTRODUÇÃO	7
2 OBJETIVOS	11
2.1 OBJETIVO GERAL	11
2.2 OBJETIVOS ESPECÍFICOS	11
3 JUSTIFICATIVA.....	12
4 FUNDAMENTAÇÃO TEÓRICA	13
4.1 CARACTERIZAÇÃO DE ESTRUTURAS MECÂNICAS	13
4.3 CINEMÁTICA DIRETA E CINEMÁTICA INVERSA.....	15
4.4 CONVERSÃO DE ATUADOR LINEAR PARA MECANISMO BIELA-MANIVELA	17
4.5 FUNDAMENTOS DE CONTROLE PID	18
5 DESENVOLVIMENTO	21
5.1 MODELO COMPUTACIONAL	21
5.1.1 GERANDO MOVIMENTO	22
5.1.2 CONTROLADOR	27
5.1.3 PLANTA.....	28
5.1.4 RESULTADOS DO MODELO	42
5.1.5 REPRESENTAÇÃO EM FLUIDSIM	48
5.2 PROTÓTIPO SIMPLIFICADO.....	49
5.2.1 MOTORES	52
5.2.2 COMUNICAÇÃO.....	54
5.2.3 SOFTWARE	57
6 CONSIDERAÇÕES FINAIS.....	61
6.1 CONCLUSÃO	61
6.2 PROJETOS FUTUROS	62
REFERÊNCIAS.....	63
APÊNDICES	66
APÊNDICE 1 - Solução de cinemática direta não linear de uma plataforma de stewart	66
APÊNDICE 2 - Solução de cinemática inversa de uma plataforma de stewart.....	84
APÊNDICE 3 – Componentes para construção do modelo simplificado.	88

APÊNDICE 4 – Software em MATLAB para simulação de modelos do protótipo..	93
Cinemática Direta (FK – Forward Kinematics).....	93
Cinemática Inversa (IK – Inverse Kinematics).....	97
Geração de trajetória (MM – Movement Maker).....	98
APÊNDICE 5 – Cmakelist	102
APÊNDICE 6 – Código em C e C++	104
Stewart Platform.cpp	104
Channel.cpp.....	119
Channel.hpp	121
Funcmoveu2d.cpp	123
USB2Dxl.hpp	125
Dynamixel.c	126
Dynamixel.h	139
Dxl_hal.c.....	142
Dxl_hal.h	149
ANEXO	151
ANEXO A – Conjunto de instruções para motores AX-12A.	151
ANEXO B – Manual do motor AX-12A.....	164
ANEXO C – Plataforma comercial tomada como base para o modelo simulado.	181

LISTA DE FIGURAS

Figura 1- Plataforma de Stewart com atuadores lineares.....	8
Figura 2 - Sistema de Biela-Manivela.....	8
Figura 3 - HEXA	9
Figura 4 - AMiBA.....	10
Figura 5 - Simulador de voo Lufthansa.	10
Figura 6 - Mecanismo Serial.	13
Figura 7 - Cadeia Fechada – Mecanismo paralelo.	14
Figura 8 - Mecanismo serial.....	17
Figura 9 - Geometria do sistema biela-manivela.....	18
Figura 10- Controlador de malha fechada.....	18
Figura 11 - Controlador em malha aberta.....	19
Figura 12 - Controlador PID.....	19
Figura 13 - Plataforma modelo.....	21
Figura 14 - Modelo em alto nível.....	22
Figura 15 - Blocos para gerar movimento.	23
Figura 16 - Cinemática inversa parte 1.1.....	23
Figura 17 - Cinemática inversa parte 1.2.....	24
Figura 18 - Cinemática inversa parte 2.....	26
Figura 19 - Cinemática inversa parte 3.....	27
Figura 20 - Contralador PID.....	28
Figura 21 - Planta.....	29
Figura 22 - Bloco de fixação.....	29
Figura 23 - Multiplexador de posição.....	30
Figura 24 - Multiplexador de velocidade.....	30
Figura 25 - Demultiplexador de força.....	30
Figura 26 - Atuador.....	31
Figura 27 - Definindo sistema de coordenadas inercial.....	32
Figura 28 - Dimensões do modelo.....	32
Figura 29 - Posições dos pontos de acoplamento.....	33
Figura 30 - Ilustração dos pontos de acoplamento na plataforma inferior.....	33
Figura 31 - Determinação dos pontos de acoplamento na plataforma inferior.....	34
Figura 32 - Ilustração dos pontos de acoplamento na plataforma superior.....	35
Figura 33 - Troca de índices do vetor de posições.....	35
Figura 34 - Pontos com relação ao sistema de coordenadas móvel.....	36
Figura 35 - Soma vetorial.....	36
Figura 36 - Operações com vetores na direção dos atuadores.....	37
Figura 37 - Direção de atuação das juntas.....	37
Figura 38 - Ilustração da direção da junta universal 1.....	38
Figura 39 - Ilustração da direção da junta universal 2.....	39
Figura 40 - Vetores dos componentes do atuador.....	39
Figura 41 - Centro de massa e ponto terminal de componentes do atuador.....	40

Figura 42 - Ilustração do atuador e seus componentes.	40
Figura 43 - Espessura dos elementos da plataforma.	41
Figura 44 - Propriedades de massa e inércia dos atuadores.	41
Figura 45 - Propriedades de massa e inércia das plataformas.	41
Figura 46 - Coeficientes do controlador PID.	42
Figura 47 - Delta de comprimento dos atuadores.	43
Figura 48 - Movimento da plataforma superior.	44
Figura 49 - Velocidade e deslocamento para movimento aplicado.	44
Figura 50 - Forças exercidas pelos atuadores.	44
Figura 51 - Sinal de entrada do controlador.	45
Figura 52 - Resposta suave ao degrau.	46
Figura 53 - Resposta abrupta ao degrau.	47
Figura 54 - Representação do circuito hidráulico.	48
Figura 55 - Representação do circuito elétrico.	49
Figura 56 - Padrão TTL dos motores.	51
Figura 57 - Protótipo construído.	51
Figura 58 - Diagrama do protótipo.	52
Figura 59 - AX-12A.	53
Figura 60 - Conversão full-duplex para half-duplex.	56
Figura 61 - Driver de comunicação.	57
Figura 62 - Erro de posição no método numérico.	58
Figura 63 - Erro de orientação no método numérico.	58
Figura 64 - Plataforma inferior.	66
Figura 65 - Plataforma superior.	67
Figura 66 - Sistema de coordenadas.	68
Figura 67 - Plataforma Inferior como triângulo equilátero.	69
Figura 68 - Altura do triângulo equilátero.	70
Figura 69 - Determinação do ponto B2.	71
Figura 70 - Triângulo formado pelos atuadores.	73
Figura 71 - Coordenadas dos pontos P_i	74
Figura 72 - Plataforma com triângulos formados por atuadores.	75
Figura 73- Possíveis pontos para $T2i$	76
Figura 74 - Retas de possíveis projeções de $T2i$	76
Figura 75 - $YT1$ em função de $XT1$	77
Figura 76 - Reescrevendo o segmento.	78
Figura 77 - Triângulo formado com a projeção do ponto $T3$	80
Figura 78 - Plataforma superior.	84
Figura 79 - Plataforma superior.	88
Figura 80 - Plataforma inferior.	89
Figura 81 - AX12.	89
Figura 82 - Conector FP04-F3.	90
Figura 83 - Technic axle 6 (black).	90
Figura 84 - Technic axle 12 (black).	91
Figura 85 - Junta universal.	91

Figura 86 - Axle to pin 43093.....	92
Figura 87 - Conector do eixo do motor.	92
Figura 88 - Cross axle extention 2M.....	92
Figura 89 - Desenho da montagem.	93

LISTA DE ABREVIATURAS

m	-Metros
m ²	-Metros cuadrados
cm	-Centímetros
mm	-Milímetros
s	-Segundos
m/s	-Metros por segundo
mm/s	-Milímetros por segundo
m ³ /s	-Metros cúbicos por segundo
kg	-Quilogramas
KPa	-Quilo pascal
N	-Newtons
V	-Volts
Mbit/s	-Mega bits por segundo
PID	-Proporcional integrativo derivativo.
TTL	-Transistor to transistor logic
USB	-Universal serial bus

1 INTRODUÇÃO

O Laboratório de Dinâmica e Simulação Veicular (LDSV) afirma que a Plataforma de Stewart foi originalmente concebido em 1965, como um simulador de aeronaves para treinamento de voo. Afirma ainda que se trata de um mecanismo paralelo que consiste de um órgão superior e uma base fixa conectadas por seis braços de atuação em pelo menos três pontos fixos. Esta configuração garante ao mecanismo seis graus de liberdade, sendo estes, 3 graus de translação e 3 graus de rotação. (USP, 2017). Outra definição apresentada por Nanua, Waldron e Murthy (1990) caracteriza a Plataforma de Stewart como um mecanismo completamente paralelo com seis graus de liberdade controlados. As juntas superiores, que acoplam os atuadores à plataforma superior, são normalmente esféricas, enquanto as juntas inferiores, que acoplam os atuadores à plataforma inferior, são normalmente universais. Segundo os autores, também pode-se utilizar duas juntas esféricas, acrescentando um grau de liberdade passivo que não afetará o comportamento do mecanismo. Estas configurações citadas resultam em mecanismos do tipo 6-SPU e 6-SPS. Os autores Rahmani, Ghanbari e Mahboobkhah (2014) utilizam em seu trabalho dois mecanismos similares aos citados anteriormente, porém as juntas que conectam os atuadores às plataformas inferior e superior, são universais, resultando em um mecanismo 6-UPU. Os autores também definem tais mecanismos como Plataformas de Stewart. Um exemplo de uma plataforma de Stewart pode ser visto na Figura 1. Liu, Lewisand e Fitzgerald (1994) destacam que, devido às restrições da sua estrutura paralela e sua cadeia cinemática fechada, a Plataforma de Stewart apresenta alta complexidade na solução do modelo cinemático direto. Outras abordagens, como a de Hersan (2015) apresentam mecanismos similares, porém com atuadores rotativos combinados ao sistema de biela-manivela, como o da Figura 2, resultando em um mecanismo 6-RSS.

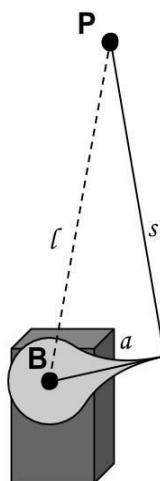
Segundo Nanua, Waldron e Murthy (1990), a Plataforma de Stewart apresenta várias vantagens quando comparada a mecanismo do tipo serial. Devido a associação em paralelo dos atuadores, os erros não se propagam junta a junta, como é o caso de mecanismo seriais, resultando em erros menores de posicionamento final. Além disso, afirmam que, como todos os atuadores estão em paralelo, eles dividem a carga, dando a Plataforma de Stewart uma capacidade de carga muito superior à dos mecanismos do tipo serial.

Figura 1- Plataforma de Stewart com atuadores lineares.



Fonte: site warsash.com.au ³

Figura 2 - Sistema de Biela-Manivela



Fonte: site instructables.com ⁴

³ Figura 1 disponível em: <http://www.warsash.com.au/suppliers/pi-physik-instrumente/index.php>, acesso 26/07/2017

⁴ Figura 2 disponível em: <http://www.instructables.com/id/Stewart-Platform/>, acesso 26/07/2017

Os autores Liu, Lewisand e Fitzgerald (1994) evidenciam o desafio da solução analítica para o modelo cinemático direto da Plataforma de Stewart, que apresenta como solução um conjunto de 30 equações algébricas não lineares que devem ser resolvidas simultaneamente, tornando sua solução analítica muito difícil e o custo computacional do cálculo numérico muito dispendioso. Outras abordagens matemáticas na resolução do modelo chegam a soluções de décima sexta ordem, mas a quantidade de coeficientes da expressão também torna seu custo computacional inviável. Opções de solução numérica, como a apresentada por Liu, Lewisand e Fitzgerald (1994), utilizando de geometrias definidas, facilitam o cálculo da cinemática direta. Outras abordagens, como a simulação apresentada por Mathworks (2002) fazem uso de controladores para atender a requisitos de posição, velocidade ou até força.

O HEXA, desenvolvido pela *Toyota Machine Works*, é um exemplo de utilização de mecanismos semelhantes à Plataforma de Stewart clássica, oferecendo seis graus de liberdade. O HEXA possui capacidade de operação em altas velocidades, desde que esteja trabalhando com baixas cargas na plataforma móvel (ROSÁRIO, 2010). O robô pode ser visto na Figura 3.

Figura 3 - HEXA



Fonte: ROSÁRIO J. M. Robótica Industrial I

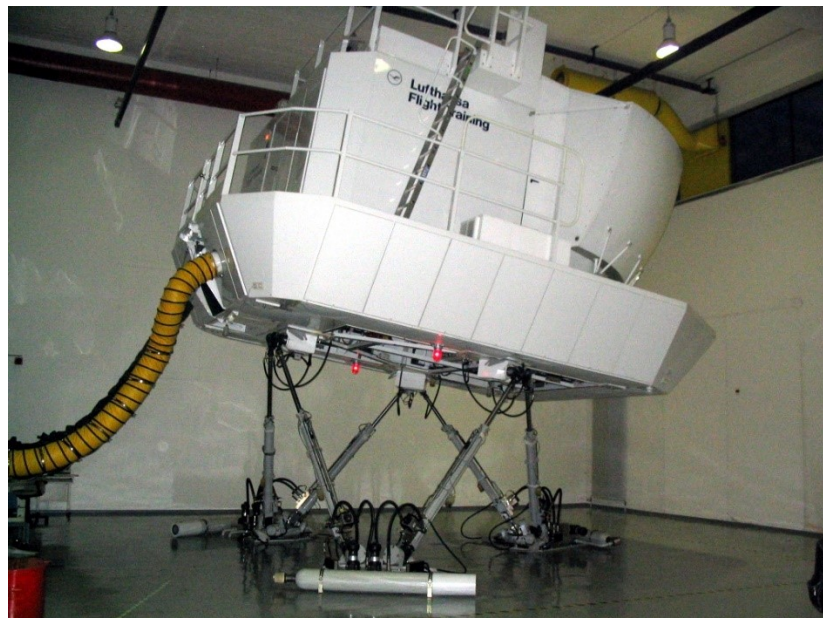
O *Array for Microwave Background Anisotropy (AMiBA)*, visto na Figura 4, é outro exemplo de utilização da plataforma de Stewart. Trata-se de um rádio telescópio, utilizado para observar ondas cósmicas. A plataforma de Stewart também é utilizada em simuladores de voo, como é o caso do simulador desenvolvido pela Lufthansa, mostrado na Figura 5.

Figura 4 – AMiBA.



Fonte: wikipedia.com⁵

Figura 5 - Simulador de voo Lufthansa.



Fonte: wikipedia.com⁶

⁵ Figura 05 disponível https://en.wikipedia.org/wiki/AMiBA#/media/File:AMiBA_1.jpg, acesso em 26/07/2017

⁶ Figura 06 disponível em: <https://upload.wikimedia.org/wikipedia/commons/3/38/Simulator-flight-compartment.jpeg>, acesso em 26/07/2017

2 OBJETIVOS

2.1 OBJETIVO GERAL

O objetivo geral deste trabalho possui duas partes distintas. A primeira parte se trata de determinar, a partir de um modelo simulado, as características dos componentes de uma Plataforma de Stewart hidráulica que atenda a requisitos de projeto definidos. A segunda, refere-se à confecção de um protótipo simplificado de uma Plataforma de Stewart, a fim de exemplificar os graus de liberdade do mecanismo.

2.2 OBJETIVOS ESPECÍFICOS

Este trabalho tem por objetivos específicos:

- O estudo de modelos cinemáticos direto e inverso para mecanismos paralelos como a Plataforma de Stewart,
- A definição da abordagem menos onerosa e sua implementação em um protótipo simplificado
- O estudo das características necessárias aos componentes de uma Plataforma de Stewart com atuadores hidráulicos, para que esta tenha mobilidade de 300mm nas três direções, capacidade de movimento angular de 60 graus de amplitude em torno dos três eixos, capacidade de carga de 20kg e velocidade máxima de 20mm/s
- A apresentação de um modelo simulado para este equipamento hidráulico.

Sendo assim, neste trabalho foram utilizadas duas abordagens diferentes. Para atender a primeira parte do objetivo, ou seja, construir um modelo computacional de forma a simular uma Plataforma de Stewart, com 300mm de amplitude de translação em todas as direções, 60 graus de rotação em torno de todos os eixos, capacidade de carga de 20kg e velocidade máxima de 20mm/s, utilizou-se ferramentas do software MATLAB®. A planta é concebida utilizando a ferramenta *SimMechanics*®, enquanto o controlador e a lógica de blocos para a geração do movimento fazem uso do *Simulink*®. Para cumprir a segunda parte do objetivo, referente ao protótipo simplificado, foi utilizado um modelo cinemático direto baseado na forma geométrica das plataformas, que não tem relação nenhuma com o modelo computacional. Neste modelo, a plataforma inferior, fixa, possui o formato de um hexágono semi-regular; enquanto a plataforma superior, móvel, possui o formato de um triângulo equilátero. Baseado nestas condições geométricas, implementou-se também um modelo cinemático inverso e uma movimentação que visa mostrar todos os seis graus de liberdade do mecanismo. A cinemática inversa e a modelagem utilizadas no protótipo também são diferentes das utilizadas no modelo computacional.

3 JUSTIFICATIVA

O desenvolvimento deste trabalho se justifica na importância do estudo acerca de mecanismos paralelos, visto que estes apresentam vantagens com relação aos mecanismos seriais, como maior capacidade de carga e melhor precisão (LIU; LEWISAND; FITZGERALD, 1994). Se justifica também devido ao fato de que um protótipo em escala é inovador para o laboratório no qual o projeto é desenvolvido (MAPL - Laboratório de Planejamento Automático de Manufatura) e colabora para a ampliação dos conhecimentos dos estudantes e da instituição a respeito do assunto tratado.

Este trabalho apresenta uma breve introdução aos conceitos da Plataforma de Stewart, direcionados aos temas tratados durante análise e desenvolvimento do projeto. Posteriormente apresenta uma fundamentação teórica acerca de conteúdos importantes à compreensão e desenvolvimento deste trabalho, como as características de estruturas mecânicas, cinemáticas direta e inversa e controladores PID. Na sequência, apresenta o desenvolvimento a análise do modelo simulado em MATLAB e seus resultados quando adaptado aos parâmetros de projetos definidos. Em seguida, apresenta o desenvolvimento do protótipo utilizando uma abordagem simplificada.

4 FUNDAMENTAÇÃO TEÓRICA

4.1 CARACTERIZAÇÃO DE ESTRUTURAS MECÂNICAS

Para definir e caracterizar as estruturas mecânicas, existem conceitos e nomenclaturas. Tais conceitos classificam as estruturas em: tipos de mecanismos, tipos de cadeias cinemáticas, tipos de juntas, como as juntas conectam os membros do mecanismo, quantos graus de liberdade o mecanismo possui, entre outros.

Quanto aos tipos de cadeias cinemáticas, elas se dividem em cadeias cinemáticas abertas e cadeias cinemáticas fechadas. As cadeias cinemáticas fechadas, como a do mecanismo da Figura 7, têm sido utilizadas desde o começo da revolução industrial, por resultar em mecanismos simples e com várias possibilidades de aplicação (PHELAN, 1962). Elas combinam múltiplas partes em um mecanismo fechado, em que a movimentação de qualquer uma das partes do mecanismo influencia o comportamento dos outros no sistema. Isto permite o controle da transferência, gerando ampliação ou atenuação de força, velocidade ou energia cinética. (ALFARO et al., 2004). Os mecanismos seriais, conforme o da Figura 6, apresentam cadeia cinemática aberta, em que os esforços sobre a estrutura, referentes ao transporte de carga do elemento terminal, são suportados em cascata pelos atuadores. Portanto, quanto mais próximo à base, maior a carga que o atuador deve suportar (SOUZA, 2007). Neste tipo de mecanismo os erros se acumulam, uma vez que o erro em uma junta anterior propagará à junta seguinte, fazendo com que esta inicie seu movimento a partir de um ponto que já possui erros associados, tendo como erro final, o seu próprio erro somado ao erro transferido da junta anterior. É um mecanismo amplamente estudado. Para grande parte das atividades industriais, esse tipo de mecanismo atende aos requisitos de precisão, tornando-o bastante popular.

Figura 6 - Mecanismo Serial.

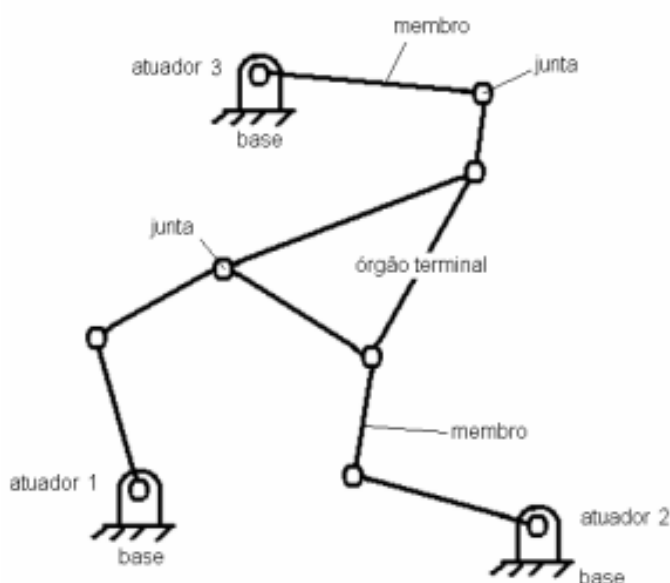


Fonte: site dreamstime.com ⁷

⁷ Figura 06 disponível em: <https://pt.dreamstime.com/imagem-de-stock-bra%C3%A7o-rob%C3%B3tico-industrial-image19546041>, acesso 26/07/2017

No que diz respeito ao tipo de estrutura, segundo Coelho (2005), uma estrutura paralela, como a Plataforma de Stewart, é um mecanismo de cadeia fechada em que o órgão terminal está conectado à base por, pelo menos, duas cadeias cinemáticas independentes. Em seu trabalho, Souza (2007), define uma cadeia cinemática como um sistema em que peças estão conectadas por juntas, podendo ser também denominadas membros ou pernas.

Figura 7 – Cadeia Fechada – Mecanismo paralelo.

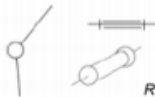
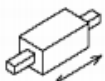
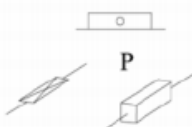
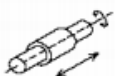
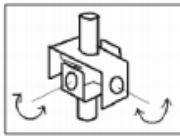
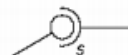


Fonte: (SOUZA, 2007)

O que caracteriza o mecanismo quanto à sua mobilidade é o número de graus de liberdade. Graus de liberdade ou DOF (Inglês, degrees of freedom) diz respeito aos movimentos possíveis de um mecanismo ou sistema (COELHO, 2005). Norton (2001) define os graus de liberdade como o número de coordenadas independentes suficientes para definir a posição e orientação de um mecanismo ou peça; ou ainda, o número de entradas necessárias para criar um movimento desejado. Dependendo do tipo de junta utilizada no mecanismo, ela pode resultar em uma restrição dos graus de liberdade. Para cadeias fechadas nos mecanismos paralelos, o grau de liberdade do elemento terminal é, quase sempre, menor que o de cada uma de suas cadeias (SOUZA, 2007).

As juntas, que unem os membros de uma estrutura mecânica, podem ser: de rotação (R), prismáticas (P), esféricas (S), helicoidais (H), cilíndricas (C) ou universais (Y), conforme Tabela 1 (SUH; RADCLIFFE, 1978; COELHO, 2005).

Tabela 1 – Tipos de juntas (SUH; RADCLIFFE, 1978; COELHO, 2005).

Junta	Geometria	Representação	Graus de Liberdade
1. de Rotação (R)			1
2. Prismática (P)			1
3. Cilíndrica (C)			2
4. Universal (U)			2
5. Esférica (S)			3

A representação da tabela é amplamente utilizada pela comunidade científica para ilustrar os movimentos possíveis de uma cadeia. A notação para as cadeias, baseada nas letras atribuídas a cada uma das juntas, está presente no trabalho de Bonev (2001) e Coelho (2005). A notação define a cadeia por uma sequência de letras, partindo da base até o órgão terminal. A junta responsável por gerar movimento, ou seja, o atuador, é apresentado como o caractere grifado. Quando o mecanismo possui mais de uma cadeia similar, um numeral é inserido à frente indicando a repetição (SOUZA, 2007). Desta forma, a Plataforma de Stewart clássica, se trata de um mecanismo com seis cadeias cinemáticas idênticas, sendo que cada uma destas possui, partindo da base até o órgão terminal, uma junta esférica, uma prismática e outra esférica, sendo que a prismática é atuada. Sendo assim, a Plataforma de Stewart clássica é classificada como um mecanismo do tipo 6 SPS.

4.3 CINEMÁTICA DIRETA E CINEMÁTICA INVERSA

Para muitas aplicações, estruturas mecânicas precisam ter seus movimentos descritos e calculados matematicamente. Por vezes, pode ser necessário conhecer a posição do elemento terminal a partir da configuração dos atuadores e juntas, ou, para uma dada posição desejada do elemento terminal, conhecer as configurações possíveis dos atuadores e juntas que a atenda. Para que isso seja possível, é preciso

descrever o comportamento das estruturas com ferramentas matemáticas, chamadas modelos cinemáticos.

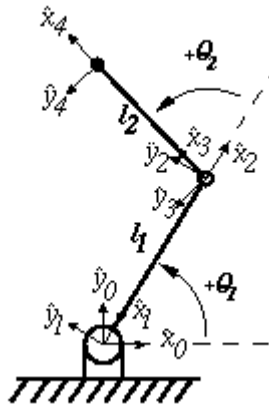
Cabral (2004) define a cinemática como o estudo da posição e da velocidade dos atuadores e ligamentos de uma estrutura. Segundo o autor, a localização de um destes ligamentos e/ou atuadores é definida pela sua posição no espaço e pela sua orientação. Para Bianchi (2011), cinemática é o estudo dos movimentos das estruturas mecânicas sem considerar suas massas ou as forças envolvidas no processo. O autor afirma que, quando se trata da cinemática, foca-se na definição da posição da estrutura.

Há dois tipos distintos de cinemática, a inversa e a direta. Na cinemática direta, o objetivo é obter a posição e a velocidade do elemento terminal para uma dada posição dos atuadores. Já na cinemática inversa, o objetivo é, dada a posição do elemento terminal, determinar as posições e velocidades dos atuadores e ligamentos. (CABRAL, 2004).

Para melhor compreensão dos conceitos, pode-se tomar como base o mecanismo da Figura 8. Trata-se de um mecanismo serial, de cadeia cinemática aberta, com dois graus de liberdade, ou seja, duas juntas articuladas geram movimento. O ponto 4, definido pelas coordenadas x_4 e y_4 , é o elemento terminal, ou seja, o ponto onde deseja-se conhecer ou determinar a localização. Os modelos cinemáticos inverso e direto buscarão determinar o comportamento do sistema com relação a esse ponto. Por exemplo: conhecendo-se os ângulos das juntas Q_1 e Q_2 , deve-se calcular o modelo cinemático direto do mecanismo para determinar a posição do elemento terminal.

Se, por outro lado, o objetivo é alcançar uma posição desejado para o elemento terminal, então é necessário encontrar a configuração para os atuadores, ou seja, os ângulos Q_1 e Q_2 , que resultem em nesta posição. Para encontrar tal configuração é preciso utilizar um modelo cinemático inverso.

Figura 8 - Mecanismo serial.



Fonte: site dca.ufrn.br⁸

4.4 CONVERSÃO DE ATUADOR LINEAR PARA MECANISMO BIELA-MANIVELA

A Plataforma de Stewart pode ser construída utilizando-se atuadores lineares ou o mecanismo biela-manivela. O mecanismo biela-manivela é utilizado para transformar movimentos de rotação em movimentos lineares (TAVARES; PINTO, 2014). Porém, os modelos cinemáticos direto e inverso comumente encontrados na literatura levam em consideração o uso de atuadores lineares. Para que estes possam ser utilizados em uma plataforma com o sistema biela-manivela, é necessário definir a relação geométrica entre os dois tipos de atuadores. A relação geométrica é apresentada nas Equações (1) e (2), baseadas no mecanismo biela-manivela da Figura 9.

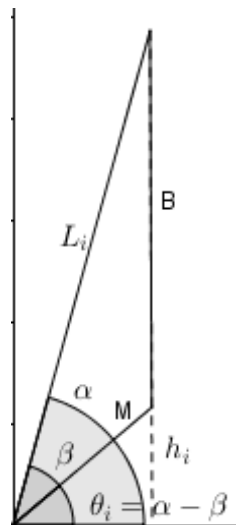
$$\theta_i = \sin^{-1}(\alpha) - \cos^{-1}(\beta) \quad (1)$$

$$\theta_i = \sin^{-1}\left(\frac{h_{i-1}}{L_i}\right) - \cos^{-1}\left(\frac{M^2 + L_i^2 - B^2}{2ML_i}\right) \quad (2)$$

Sendo que " θ_i " é o ângulo do atuador rotacional, L_i o comprimento do atuador linear equivalente, "M" e "B" são os comprimentos da manivela e da biela, respectivamente. A relação é dada pela Lei dos Cossenos aplicada sobre o triângulo da Figura 9.

⁸ Figura 08 disponível em: <http://www.dca.ufrn.br/~lmarcos/courses/robotica/exercicios/ex1.html>, acesso 26/07/2017

Figura 9 - Geometria do sistema biela-manivela.



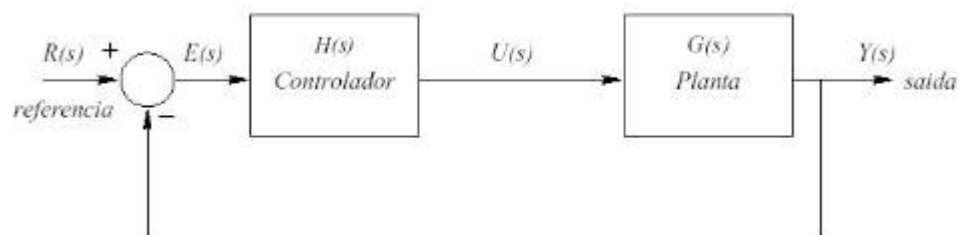
Fonte: elaborado pelo autor.

4.5 FUNDAMENTOS DE CONTROLE PID

O controlador tem por objetivo controlar algo, como por exemplo, o nível de um tanque, a posição, a velocidade e/ou a aceleração de um mecanismo, a temperatura em uma caldeira, entre outros sistemas. A estes sistemas controlados, dá-se o nome de planta (OGATA, 2005).

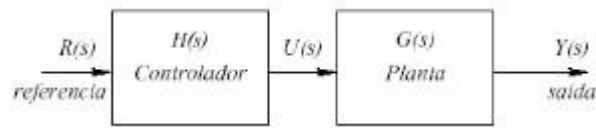
O controle trata-se de um componente que, dado uma condição de referência desejada para a planta, atua sobre ela de forma a fazê-la atender esta condição, podendo ou não levar em consideração a condição atual em que a planta se encontra. Se a ação de controle leva em consideração a condição atual da planta e, portanto, age a partir de um erro entre a condição atual e a de referência, diz-se que este controlador é de malha fechada. Se a ação de controle não leva tal condição em consideração, diz-se que o controle é de malha aberta (OGATA, 2005). Os diagramas para controle de malha fechada e malha aberta são apresentados nas Figura 10 e Figura 11, respectivamente.

Figura 10- Controlador de malha fechada.



Fonte: elaborada pelo autor.

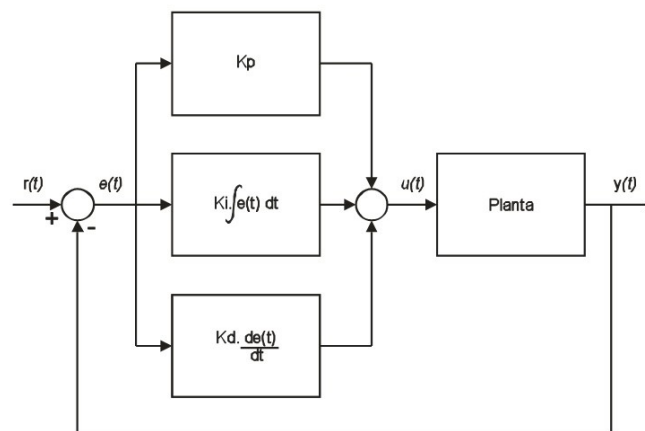
Figura 11 - Controlador em malha aberta.



Fonte: elaborada pelo autor.

Controladores de malha fechada um pouco mais complexos agem não somente sobre o sinal do erro, mas também com base na derivada do erro e na integral do erro. A estes chama-se PID (OGATA, 2005), conforme o da Figura 12.

Figura 12 - Controlador PID.



Fonte: elaborada pelo autor.

Segundo Ogata (2005), em um primeiro instante, quando um sinal de referência é aplicado, e o erro entre este e a condição atual é obtido, entende-se que neste momento 0% do erro foi compensado pelo controlador. Conforme o sistema age sobre o erro, tornando-o menor, passa-se por um regime de transição, em que as variações do erro são expressivas, chamado regime transitório. Quando grande parte deste erro já foi eliminado e, a partir de então, a variação do erro é pequena, menor que 2%, diz-se estar em um regime permanente.

A parcela proporcional, explica Ogata (2005), age sobre a diferença entre a posição de referência e a posição atual, sendo que sua ação tem influência sobre os regimes transitório e permanente. Porém, sua atuação é proporcional, como o próprio nome sugere, ou seja, quando o erro é grande sua contribuição é grande, quando o erro é pequeno sua contribuição é pequena. A parcela proporcional é representada por K_p na Figura 12.

Analisando-se agora a derivada do erro, tem-se uma noção sobre a variação do erro no tempo. Segundo Ogata (2005), no PID pode-se agir sobre esta variação por meio do coeficiente K_d , chamado de derivativo, fazendo com que a planta responda bruscamente ou lentamente, ou seja, gerando uma agindo na taxa de compensação do erro no tempo. Nota-se que a parcela derivativa tem maior influência

no regime permanente, em que o erro é mais expressivo. É importante salientar que as duas abordagens têm suas consequências, sendo que existe um meio termo ótimo entre elas (OGATA, 2005). Se o erro diminui drasticamente, significa que, para um pequeno sinal de erro, a ação da parcela derivativa é muito expressiva, e sendo assim, em uma situação em que o erro seja realmente pequeno, pode ser que a ação sobre o erro seja maior do que a necessária, fazendo com que o sistema passe do ponto de referência. A este comportamento dá-se o nome de *overshoot* (OGATA, 2005). Por outro lado, uma resposta lenta ao erro, o que acontece no caso em que a ação da parcela derivativa é pouco expressiva, pode não ser adequado, principalmente em casos em que o sistema precisa de um tempo de resposta relativamente rápido.

Por fim, Ogata (2005) afirma em seu livro que a parcela integrativa, representada pelo coeficiente K_i , age sobre a soma dos erros no tempo, influenciando sobre ambos os regimes, transitório e permanente, tornando-se então maior conforme o tempo passa. Mesmo que o erro entre dois dados instantes seja pequeno, sua ação ainda se torna mais expressiva devido a soma deste erro aos demais. Quando o erro se torna zero, nada mais é somado, e esta parcela não mais influencia no sinal de saída do controlador.

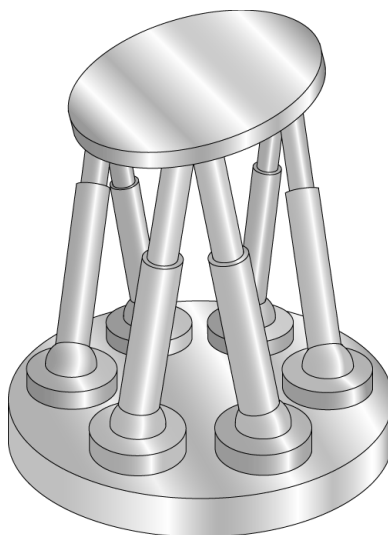
Como a ação do controlador sobre a planta é a soma destas três ações, encontrar o ponto de equilíbrio entre os coeficientes é o desafio para garantir uma boa atuação do controlador.

5 DESENVOLVIMENTO

5.1 MODELO COMPUTACIONAL

O modelo busca simular uma planta de uma Plataforma de Stewart real, construída a partir das ferramentas *Simechanics* e *Simulink*, buscando controlar seus movimentos a partir da ação de um controlador PID. Todo este desenvolvimento foi realizado tomando como base o modelo já existente, disponível para *download* no link da referência [26]. O ponto de partida para a compreensão do mesmo foi o conteúdo presente nos links das referências [11] e [12]. O modelo é baseado em uma plataforma conforme a da Figura 13.

Figura 13 - Plataforma modelo.

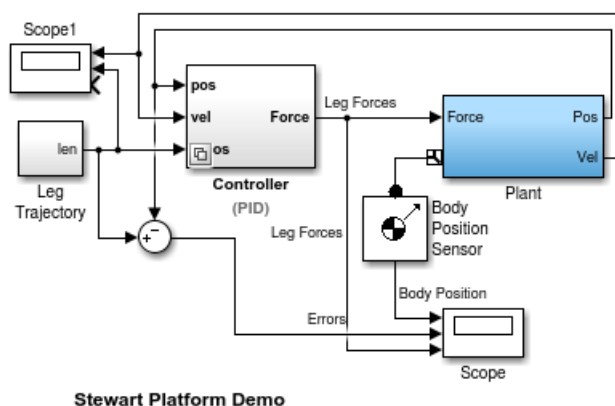


Fonte: site mathworks.com⁹

De forma geral, o modelo parte de um movimento desejado, chamado “Leg Trajectory”, aplicado a entrada em um controlador PID, que recebe informações de *feedback* da planta, ou seja, da própria plataforma. A plataforma é munida de sensores de deslocamento e velocidade em cada atuador. Um sensor de posição da plataforma superior, chamado “Body Position Sensor”, permite analisar se o movimento desejado está efetivamente sendo realizado. Dadas as informações de entrada ao controlador, ele age sobre a força de cada um dos atuadores buscando alcançar a posição desejada. A Figura 14 mostra o controlador no *Simulink*.

⁹ Figura 13 disponível em: <https://www.mathworks.com/company/newsletters/articles/creating-a-stewart-platform-model-using-simmechanics.html?requestedDomain=www.mathworks.com>, acesso 26/07/2017

Figura 14 - Modelo em alto nível.



Fonte: site www.mathworks.com¹⁰

Os “Scope” e “Scope1” são visualizadores gráficos. O “Scope” permite analisar os valores da posição da plataforma superior, a força em cada atuador e o erro entre a posição desejada e a atual. O “Scope1” fornece informações a respeito da posição e da velocidade de cada atuador.

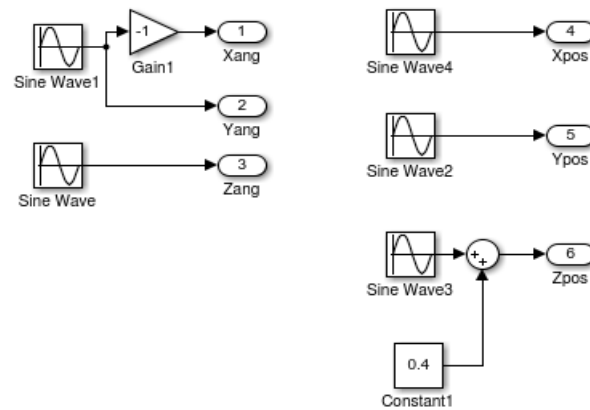
5.1.1 GERANDO MOVIMENTO

Vários tipos de movimentos podem ser gerados, dependendo dos blocos utilizados para gerar a trajetória na ferramenta *Simulink*. Para esta aplicação trabalhou-se apenas com funções senoidais e degrau. Os movimentos foram gerados como visto na Figura 15.

As saídas de 1 a 3 representam os movimentos angulares, enquanto as saídas 4 a 6 representam as translações. Ao movimento na direção Z soma-se uma constante, referente a altura inicial da plataforma superior. Estes valores alimentam uma segunda parte do modelo, visto nas Figuras 16 e 17. Na verdade, as figuras representam um mesmo diagrama de bloco, sendo que a Figura 17 é a continuação do diagrama da Figura 16 após o bloco “Compute vector of leg lengths”.

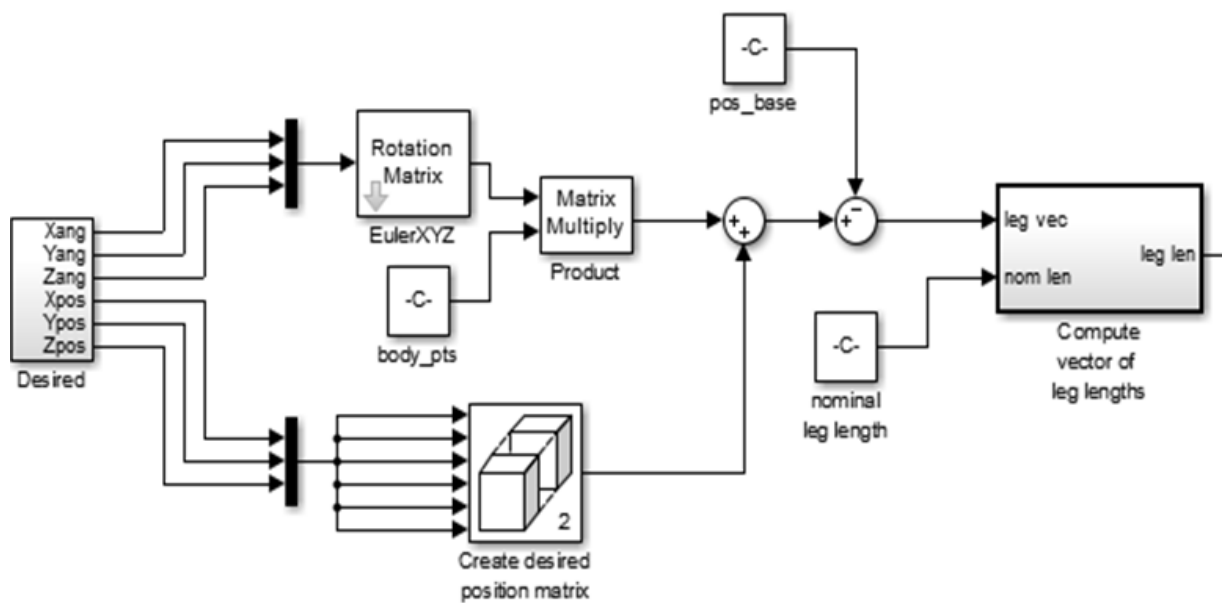
¹⁰ Figura 14 disponível em: <http://www.mathworks.com/help/physmod/sm/examples/stewart-platform-with-control-reference-trajectory.html>, acesso 26/07/2017

Figura 15 - Blocos para gerar movimento.



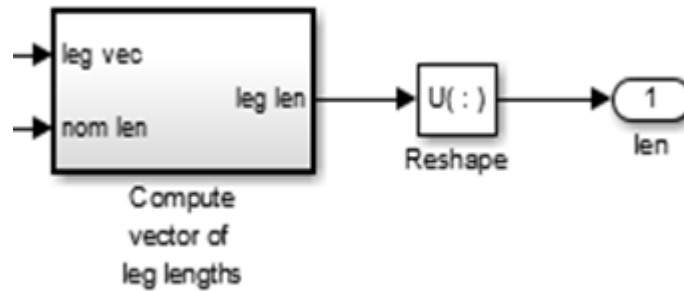
Fonte: simulador da plataforma no Simulink.

Figura 16 - Cinemática inversa parte 1.1



Fonte: simulador da plataforma no Simulink.

Figura 17 - Cinemática inversa parte 1.2.



Fonte: simulador da plataforma no Simulink.

Para o cálculo da cinemática inversa do modelo, leva-se em consideração duas posições da plataforma superior. A primeira posição, chamada de agora em diante de posição de partida e representada pela letra “n”, é a posição de repouso da plataforma, em que as plataformas superior e inferior se encontram paralelas e concêntricas, de forma que todos os atuadores tenham o mesmo comprimento, chamado de agora em diante de comprimento de partida e representado por “ l_n ”. A segunda posição, chamada de agora em diante de posição instantânea e representada pela letra “i”, é a posição da plataforma no instante, calculada pela cinemática inversa, com base no movimento desejado aplicado à plataforma móvel. Por exemplo: caso o movimento desejado para a plataforma móvel seja uma senóide, a plataforma precisará passar por um conjunto de pontos de forma a descrever o movimento. A plataforma passará por cada um destes pontos em um dado instante de tempo, dependendo da velocidade do movimento desejado. Então, para cada instante de tempo há uma posição que a plataforma móvel deve alcançar, e consequentemente, um comprimento para cada um dos atuadores quem fazem com que a plataforma móvel alcance a posição desejada. Esta é a configuração instantânea da plataforma, e estes são os comprimentos instantâneos dos atuadores. A lógica desta parte do modelo baseia-se na Equação (3):

$$\left| (R * p_t + p) - p_b \right| - l_n = \Delta l_i \quad (3)$$

Para melhor compreensão, imagine que a Equação (3) exemplifica o cálculo da cinemática inversa para um único atuador. O cálculo toma como base dois sistemas de coordenadas, um móvel e um fixo. O sistema de coordenadas fixo, inercial, está posicionado no centro da plataforma inferior. O sistema de coordenadas móvel está posicionado no centro da plataforma superior.

Para compreender o que cada um dos termos da Equação (3) significa, imagine que a plataforma superior tenha transladado, o que significa dizer que o sistema de coordenadas móvel transladou X_{pos} , Y_{pos} e Z_{pos} . Nestas condições, “p” é então o

vetor 3x1 que representa a posição do sistema de coordenadas móvel em relação ao sistema de coordenadas inercial.

Por sua vez, " p_t " é o vetor 3x1 contendo as coordenadas do ponto em que o atuador está acoplado na plataforma superior, referenciado no sistema de coordenadas móvel, " p_b " é o vetor 3x1 contendo as coordenadas do ponto onde o atuador está acoplado na plataforma inferior, referenciado no sistema de coordenadas inercial e, conforme já dito, " l_n " é o comprimento do atuador na posição de partida. Por fim, "R" representa a matriz 3x3 de rotações de Euler, contendo os ângulos de rotação do sistema de coordenadas móvel *Xang*, *Yang* e *Zang*, em relação ao sistema de coordenadas inercial.

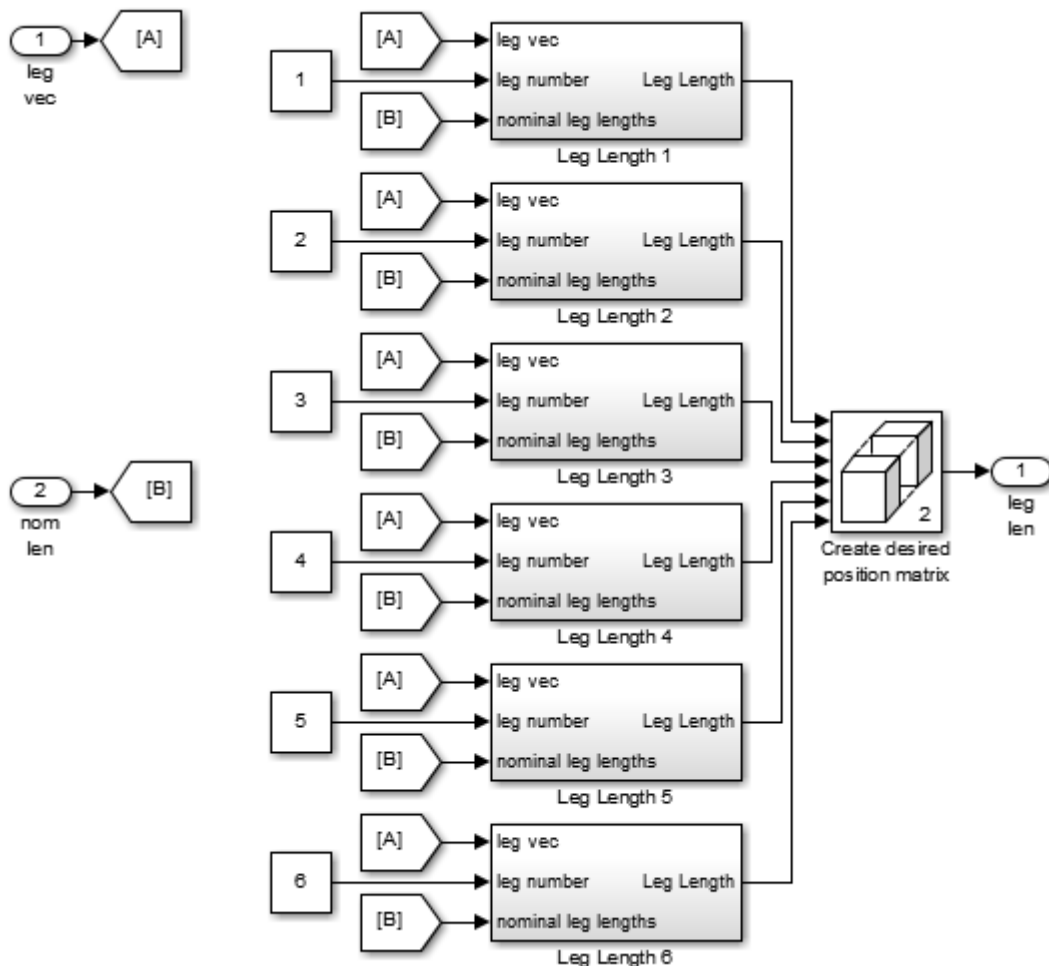
Sendo assim, multiplicando o ponto de acoplamento da plataforma superior, referenciado no sistema de coordenadas móvel, em sua posição de partida (p_t), pela matriz de rotação do sistema de coordenadas móvel (R), obtém-se um vetor 3x1 com as coordenadas deste ponto após as rotações da plataforma superior, referenciado no sistema de coordenadas móvel. Somando o valor deste ponto à posição do próprio sistema de coordenadas móvel em relação ao sistema de coordenadas inercial (p), obtém-se um vetor 3x1 com as coordenadas do ponto em relação ao sistema de coordenadas inercial, ou seja, as coordenadas do ponto de acoplamento do atuador à plataforma superior com relação ao sistema de coordenadas inercial. Uma vez que as coordenadas dos pontos de acoplamento do atuador nas duas plataformas estão definidos com relação ao mesmo referencial, subtraindo-se as coordenadas do ponto de acoplamento do atuador à plataforma superior das coordenadas do ponto de acoplamento do atuador à plataforma inferior, obtém-se um vetor 3x1 com mesmo módulo e direção que o atuador, representando então o atuador na posição instantânea. Tirando-se o módulo deste vetor, obtém-se o valor do comprimento do atuador na posição instantânea, ou seja, após as translações (X_{pos} , Y_{pos} e Z_{pos}) e rotações (*Xang*, *Yang* e *Zang*) determinadas. Subtraindo-se este comprimento do comprimento do atuador na posição de partida, tem-se a variação do comprimento do atuador, Δl_i , ao realizar o movimento desejado

No *Simulink*, os cálculos do delta do comprimento dos atuadores são realizados todos ao mesmo tempo pelo diagrama de blocos. Os vetores são transformados em matrizes 3x6 contendo os dados de todos os atuadores, mas para cada um dos atuadores separadamente o cálculo é realizado conforme descrito no parágrafo anterior. Desta forma, observando-se o modelo em *Simulink*. Os ângulos gerados *Xang*, *Yang* e *Zang* são arranjado segundo uma matriz de Euler 3x3, como dito anteriormente, representada por "R". Os valores contidos na matriz 3x6 "body_pts" são referentes aos pontos de acoplamento dos atuadores na plataforma superior, representado por " p_t ". As translações X_{pos} , Y_{pos} e Z_{pos} são agrupadas em uma matriz de translação 3x6, representada por "p" na equação, que ao ser somada à matriz 3x6 de rotação formada pela multiplicação " $(R * p_T)$ " resulta em uma matriz 3x6 de transformação de coordenadas " $(R * p_T + p)$ ", ou seja, resulta na combinação entre rotação e translação dos pontos de acoplamento da plataforma superior com relação

ao sistema de coordenadas inercial. Em seguida, estes pontos são subtraídos dos pontos de acoplamento dos atuadores na plataforma inferior, salvos na matriz 3x6 “pos_base” e representados por “ p_b ” na equação, resultando na parte “ $(R * p_T + p) - p_b$ ” da equação, ou seja, nos vetores que representam os atuadores.

O comprimento original dos atuadores, salvo na matriz 1x6 “leg_lenght” e representa por l_n na equação, é subtraído dos módulos “ $|(R * p_T) - p_b|$ ”, calculados separadamente para cada um dos atuadores, resultando na forma final da Equação (3). A operação do módulo é executada pelo bloco “Compute vector of leg lenght”, e é feita atuador a atuador, se tratando do módulo de cada um dos vetores que representam os atuadores e compõem a matriz 3x6 resultante da operação “ $(R * p_T + p) - p_b$ ”, não é então referente ao módulo desta matriz 3x6. Este bloco pode ser visto em detalhes na Figura 18.

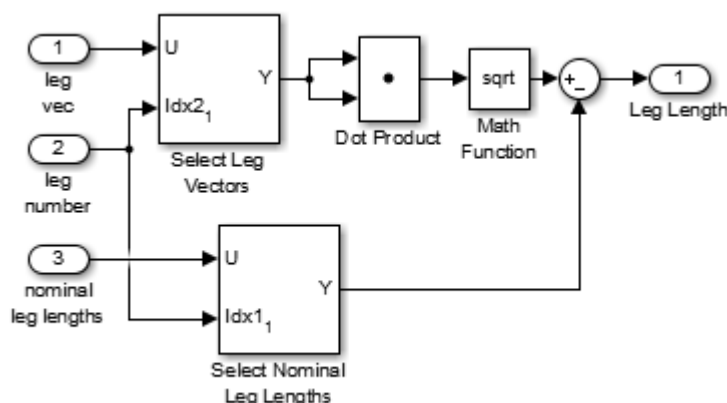
Figura 18 - Cinemática inversa parte 2.



Fonte: simulador da plataforma no Simulink.

Após o cálculo do módulo, cada valor é adicionado à matriz de posição desejada, que será a saída do bloco. Expandindo o bloco “Leg Lenght1” para compreender como a operação é feita, observa-se o que é apresentado na Figura 19.

Figura 19 - Cinemática inversa parte 3.



Fonte: simulador da plataforma no Simulink.

O bloco “Select Leg Vector” tem a função de restringir a entrada do vetor “leg vec”, de 6 elementos, ao elemento de índice “ $idx2_1$ ”, referente ao atuador que deseja-se calcular o comprimento. No exemplo da Figura 19, o bloco “Leg Length1” faz o cálculo do atuador número 1. Em outras palavras, a entrada é o índice “ $idx2_1$ ” do vetor “leg vec”, índice este que, na verdade, é o número do atuador representado por “leg number”, limitando assim a entrada ao elemento 1 do vetor “leg vec”. O bloco “Select Nominal Leg Lengths” tem função similar, porém é referente ao vetor “nominal leg lengths”, dos valores de comprimento nominal dos atuadores. Sendo assim, para cada atuador realiza-se o produto escalar do vetor que o representa por ele mesmo. Como resultado, obtém-se todos seus componentes ao quadrado. Retirando-se a raiz deste resultado, com o bloco “sqrt”, obtém-se o módulo deste vetor. Ao ser subtraído de seu comprimento original advindo do bloco “Select Nominal Leg Lengths”, resulta no delta de comprimento deste atuador em específico para que o movimento desejado possa ser realizado.

Voltando à Figura 17, este vetor coluna resultante das operações com os vetores dos atuadores é convertido em um vetor linha pelo bloco “Reshape” e, enfim, transformado no vetor “len” de saída do bloco, que servirá como entrada em outra parte do modelo.

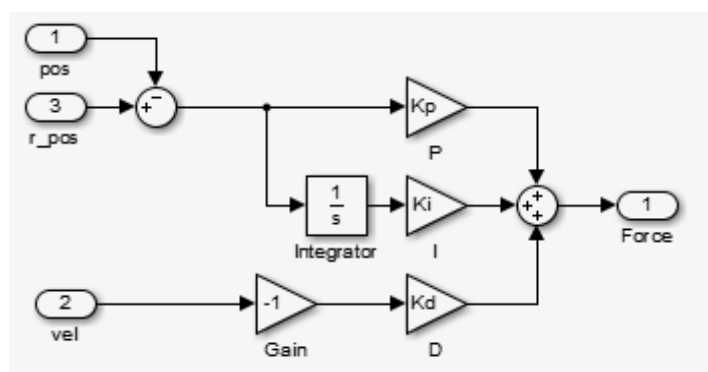
5.1.2 CONTROLADOR

Para que a planta da plataforma de Stewart realize o movimento desejado sem a necessidade de um modelo cinemático direto verificando online a precisão do movimento, é necessário que se implemente um controle de posição. Esse controle pode ser feito para velocidade e aceleração também, mas exigiria um controlador mais complexo do que o que será discutido.

Na planta, pode-se agir na pressão dos atuadores, o que significa atuar na força

que o atuador aplica sobre a plataforma. Sendo assim, tem-se um problema de controle definido. Variando-se a força conforme for conveniente, mantém-se um controle sobre a posição real visando atingir a posição desejada. Para tal, utilizou-se o controlador PID que, conforme pode ser visto na Figura 20, age sobre a força aplicada pelo atuador, baseado nos *feedbacks* de velocidade e posição.

Figura 20 - Contralador PID.



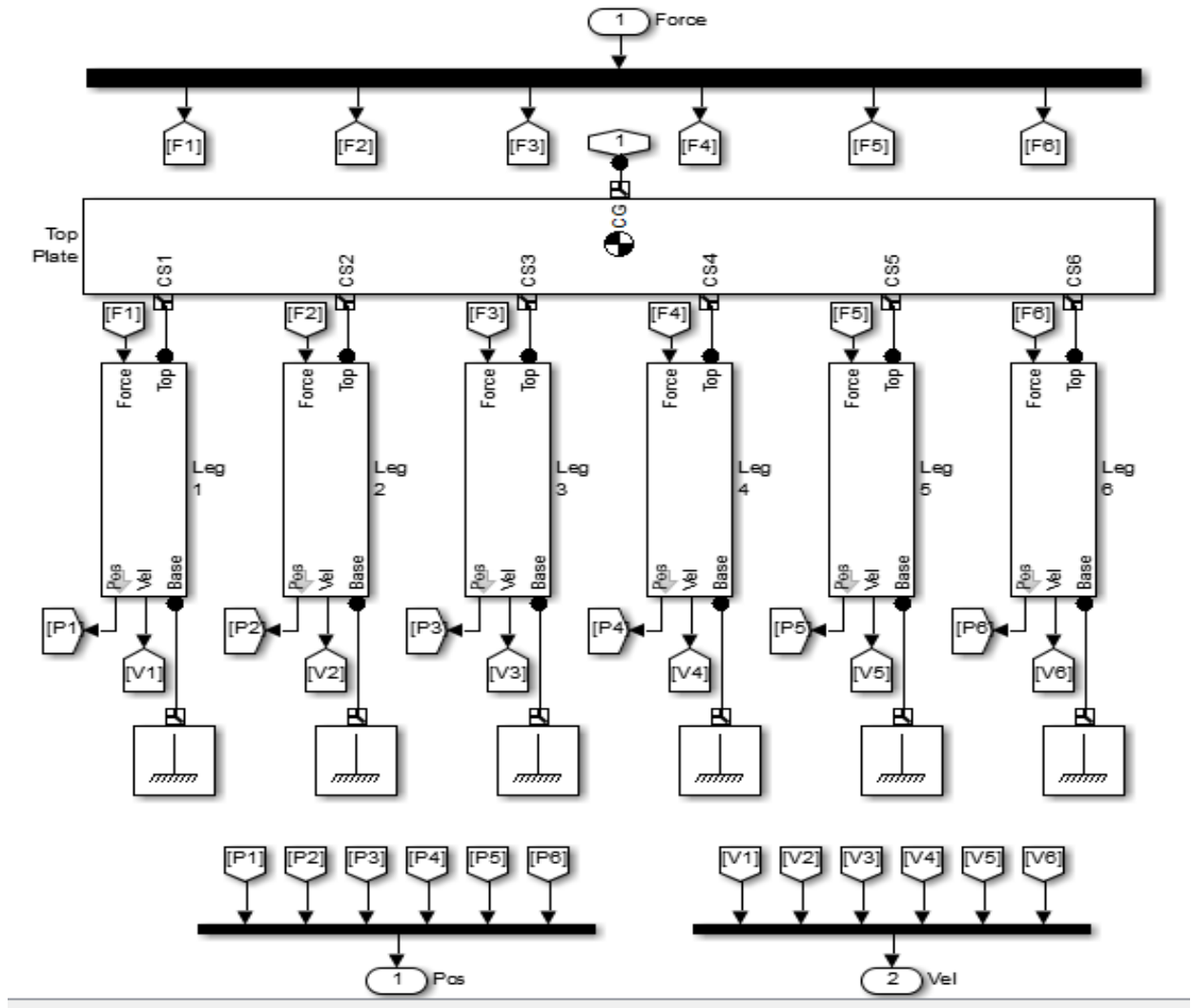
Fonte: simulador da plataforma no Simulink.

Para o mecanismo modelado em *simulink*, o controlador recebe como entrada o vetor de deltas entre o comprimento desejado e o lido para cada atuador, ou seja, o erro de posição. Recebe também a velocidade do deslocamento, ou seja, a derivada do erro. A estes aplicam os ganhos K_i , K_p e K_d , e somando-os, obtém-se o vetor com os valores das forças necessárias a cada atuador para corrigir a diferença entre valor de posição lido e o valor desejado.

5.1.3 PLANTA

A planta é o elemento controlado. Neste caso, a planta é a própria Plataforma de Stewart, composta por uma plataforma inferior cilíndrica, uma superior cilíndrica e atuadores pneumáticos cilíndricos (sendo o cilindro interno maciço e o externo oco), conectados às plataformas por juntas universais. O modelo criado no *Simechanics* se assemelha ao apresentado na Figura 21.

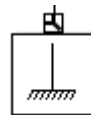
Figura 21 - Planta.



Fonte: simulador da plataforma no Simulink.

Cada um dos atuadores (chamados “Leg”) está conectado a um bloco de fixação como o da Figura 22, o que significa que todos estão presos a uma base imóvel. Os atuadores também estão conectados a uma plataforma superior, chamada “Top Plate”, cujas características são definidas em código e serão exploradas adiante.

Figura 22 - Bloco de fixação.

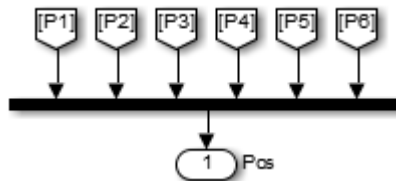


Fonte: simulador da plataforma no Simulink.

As barras pretas mostradas nas Figuras 23 a 24 são dois multiplexadores e um demultiplexador, respectivamente. O multiplexador da Figura 23 cria um vetor chamado “pos” com os valores [P1], [P2], [P3], [P4], [P5] e [P6], advindos do sensor de posição. O multiplexador da Figura 24 cria um vetor “vel” com os valores [V1], [V2],

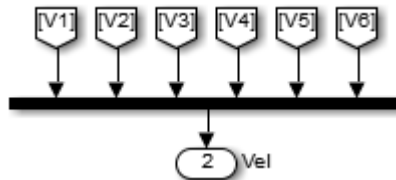
[V3], [V4], [V5] e [V6], advindos do sensor de velocidades. Ambos os vetores serão utilizados no *feedback* do controlador PID. Por fim, o demultiplexador transforma um vetor “Force” em seis valores de força, [F1], [F2], [F3], [F4], [F5] e [F6], que são a resposta do controlador PID, e serão aplicadas pelos atuadores à plataforma superior. O demultiplexador pode ser visto na Figura 25.

Figura 23 - Multiplexador de posição.



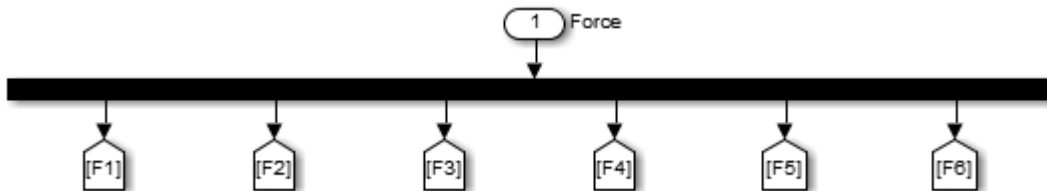
Fonte: simulador da plataforma no Simulink.

Figura 24 - Multiplexador de velocidade.



Fonte: simulador da plataforma no Simulink.

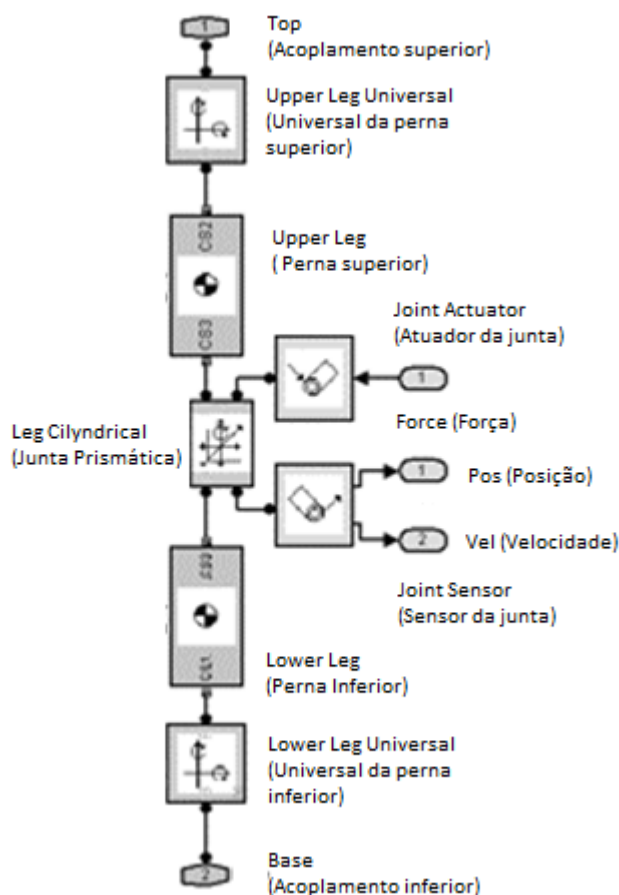
Figura 25 - Demultiplexador de força.



Fonte: simulador da plataforma no Simulink.

Quanto aos atuadores, o bloco que os representa pode ser visto com detalhes na Figura 26.

Figura 26 – Atuador.



Fonte: site [mathworks.com](https://www.mathworks.com)¹¹

O atuador possui um rótulo à base, chamado “*base*”, uma junta universal que conecta a base à parte inferior do atuador, chamada “*Lower Leg Universal*”. A parte inferior do atuador trata-se de um cilindro de parede fina, chamado “*Lower Leg*”, cujas características físicas serão configuradas em código. O mesmo é válido para a parte superior do atuador, composto por um cilindro maciço chamado “*Upper Leg*”. As duas partes estão conectadas por uma junta cilíndrica chamada “*Leg Cylindrical*”, que permite a translação do cilindro maciço no interior do cilindro oco, formando um pistão. À estas juntas estão associados um atuador de juntas chamado “*Joint Actuator*”, que age na junta de forma a aplicar uma força que gere movimento, e um sensor chamado “*Joint Sensor*”, responsável por fazer leituras do estado da junta. Neste caso, os sensores são de posição e velocidade (conforme já comentados anteriormente), e são responsáveis por fornecer *feedback* ao controlador PID. Por fim, uma junta universal chamada “*Upper Leg Universal*”, une “*Upper Leg*” à plataforma superior, criando outro rótulo chamado “*Top*”.

Uma vez construído o modelo, ele precisa ser configurado. Para tal, define-se

¹¹ Figura 26 disponível em: <https://www.mathworks.com/company/newsletters/articles/creating-a-stewart-platform-model-using-simmechanics.html?requestedDomain=www.mathworks.com>, acesso 26/07/2017

algumas variáveis que serão utilizadas para caracterizar cada uma das partes da plataforma. O primeiro passo a ser feito é estabelecer os sistemas de coordenadas tomados como referência para o movimento. Apesar de ter-se conhecimento de onde eles se localizam conceitualmente, é necessário informar ao *software* a localização. Utilizando a linguagem de programação do MATLAB, define-se a posição do sistema de coordenadas inercial, como visto na Figura 27.

Figura 27 - Definindo sistema de coordenadas inercial.

```
deg2rad = pi/180;
x_axis = [1 0 0];
y_axis = [0 1 0];
z_axis = [0 0 1];
```

Fonte: código “.m” do simulador.

Em seguida, no trecho de código da Figura 28, cria-se os vetores dos pontos em que os atuadores estão acoplados às plataformas, chamados por “pos_base” e “pos_top”. Além destes, define-se duas constantes, que serão utilizadas posteriormente como ferramenta para o cálculo de tais pontos. São estas “alpha_b” e “alpha_t”. Declara-se também as dimensões da plataforma superior, como “height”, que representa a altura da plataforma superior quando em repouso, “radius_b”, referente ao raio da plataforma inferior e “radius_t”, referente ao raio da plataforma superior. Estes últimos três foram definidos com base em uma plataforma comercial, tomada como parâmetro para estabelecer os pré-requisitos do modelo simulado. O *datasheet* desta plataforma pode ser encontrado no Anexo C.

Figura 28 - Dimensões do modelo.

```
pos_base = [];
pos_top = [];
alpha_b = 2.5*deg2rad; % +- offset angle from 120 degree spacing on base
alpha_t = 10*deg2rad; % +- offset angle from 120 degree spacing on top
height = 0.4; % 0.4 meter height in home configuration
radius_b = 0.35; % base radius in meters
radius_t = 0.2; % top radius in meters
```

Fonte: código “.m” do simulador.

Uma vez criados os vetores de posição, define-se então as posições, conforme Figura 29.

Figura 29 - Posições dos pontos de acoplamento.

```

for i = 1:3,
    % base points
    angle_m_b = (2*pi/3) * (i-1) - alpha_b;
    angle_p_b = (2*pi/3) * (i-1) + alpha_b;
    pos_base(2*i-1,:) = radius_b * [cos(angle_m_b), sin(angle_m_b), 0.0];
    pos_base(2*i,:) = radius_b * [cos(angle_p_b), sin(angle_p_b), 0.0];

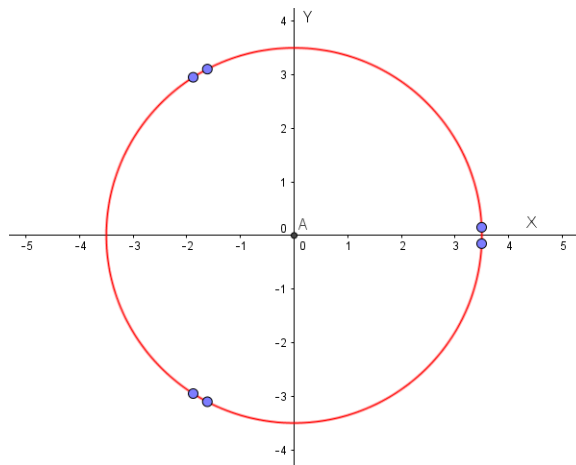
    % top points
    % Top points are 60 degrees offset
    angle_m_t = (2*pi/3) * (i-1) - alpha_t + 2*pi/6;
    angle_p_t = (2*pi/3) * (i-1) + alpha_t + 2*pi/6;
    pos_top(2*i-1,:) = radius_t * [cos(angle_m_t), sin(angle_m_t), height/radius_t];
    pos_top(2*i,:) = radius_t * [cos(angle_p_t), sin(angle_p_t), height/radius_t];
end

```

Fonte: código “.m” do simulador.

Sendo “ α_b ” definido como 2,5 graus, como visto na Figura 29, conforme o “i” varia, “ angle_m_b ” resultará em: -2,5, 117,5 e 237,5 graus. Por outro lado, “ α_p ” resultará em: +2,5, 122,5 e 242,5. Estes ângulos representam a posição sobre o raio em que cada um dos atuadores está acoplado à plataforma inferior, sendo que o eixo X no sentido positivo representa o 0 graus, ou seja, os atuadores estão acoplados à plataforma inferior conforme mostrado na Figura 30.

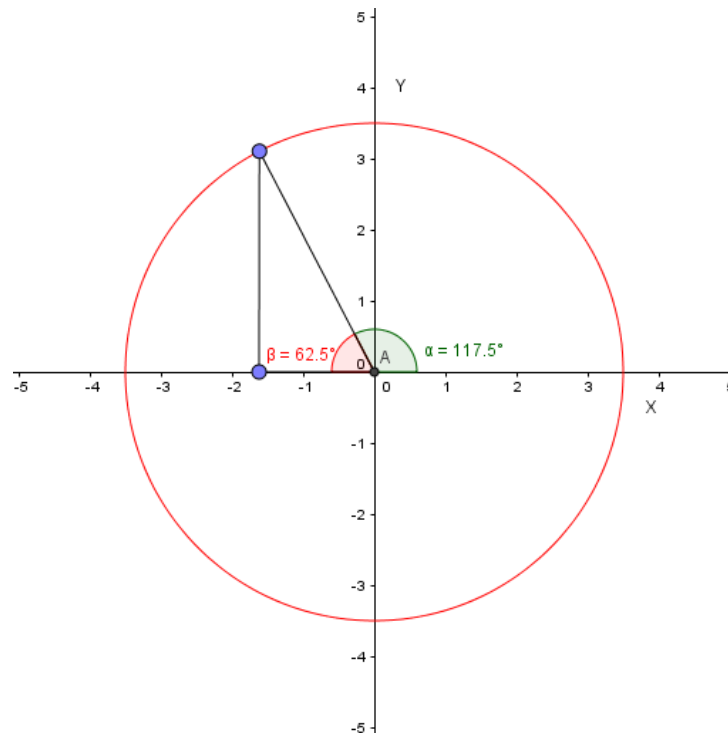
Figura 30 - Ilustração dos pontos de acoplamento na plataforma inferior.



Fonte: elaborado pelo autor.

Para determinar as coordenadas X e Y faz-se uso de relações trigonométricas que podem ser analisadas a partir de triângulos conforme o apresentado na Figura 31 para o ângulo de 117,5 graus.

Figura 31 - Determinação dos pontos de acoplamento na plataforma inferior.

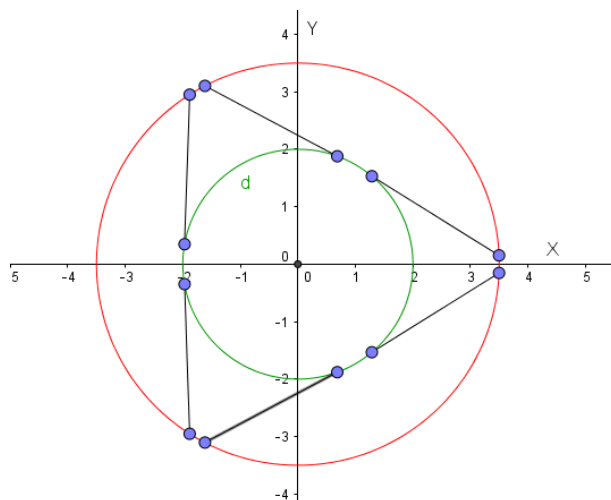


Fonte: elaborado pelo autor.

A coordenada X de cada um dos pontos é dada pelo raio multiplicado pelo cosseno do ângulo em verde, assim como a coordenada Y é dada pelo raio multiplicado pelo seno deste mesmo ângulo. Os sinais dos valores das funções seno e cosseno variam de acordo com os quadrantes, por exemplo: o seno de 62,5 graus tem o mesmo valor e sinal do seno de 117,5 graus, já o cosseno de 117,5 graus tem o mesmo valor do cosseno de 62,5 graus, porém é negativo. Conforme visto na Figura 31, o valor da coordenada X é negativa, então ao fazer o cosseno de 117,5 graus, além de encontrar um valor equivalente ao do cateto adjacente ao ângulo de 62,5 graus, este valor já é obtido com o sinal correto da coordenada X , neste caso, negativo. A coordenada Z dos pontos da base inferior é 0, uma vez que estão no plano XY . Além disso, nota-se que cada par de pontos é separado por 5 graus, enquanto que os ângulos que se encontram no meio destes pares estão espaçados de 120 graus. São eles 0, 120 e 240 graus.

Quanto aos pontos da plataforma superior, uma vez que “ α_t ” foi definido como 10 graus, conforme o valor de “ i ” se altera, os valores de “ α_m ” resultarão em: 50, 170 e 290, enquanto os ângulos de “ α_p ” resultarão em: 70, 190 e 310. Ou seja, os atuadores estão acoplados à plataforma superior conforme mostrado na Figura 32, que apresenta a vista superior da Plataforma de Stewart, sendo que o círculo verde representa a plataforma superior, e o círculo vermelho representa a plataforma inferior.

Figura 32 - Ilustração dos pontos de acoplamento na plataforma superior.



Fonte: elaborado pelo autor.

O eixo X, em 0 graus, divide ao meio os pontos a -2,5 graus e 2,5 graus da plataforma inferior. Já o eixo que divide ao meio os pontos 50 e 70 graus da plataforma superior, está a 60 graus. O mesmo acontece para qualquer par conseguinte de pontos das plataformas, ou seja, as plataformas estão defasadas em 60 graus.

Conforme analogia feita para a plataforma inferior, os pontos X e Y podem ser dados pelas relações do raio com os senos e cossenos dos ângulos, mas neste caso, a coordenada Z é diferente de 0. A coordenada Z dos pontos da plataforma superior é igual à altura inicial definida para o estado de repouso, ou seja, vale 0.4 metros para este modelo. Para facilitar a construção do código, multiplica-se todos os elementos do vetor posição pelo raio de uma só vez e, no caso da coordenada Z, divide-se novamente pelo raio, de forma que a operação não gere nenhum resultado e portanto, o resultado se mantenha em 0.4.

Conforme foi definido pela parte do código da Figura 29 e visto na Figura 32, o ponto 1 da plataforma inferior, em -2,5 graus, está conectado ao ponto 6 da plataforma superior, em 310 graus. Para que os índices sejam equivalentes, troca-se as posições, de forma que o ponto 1 da plataforma superior esteja ligado ao ponto 1 da plataforma inferior, e assim por diante. Tal permutação é vista na Figura 33.

Figura 33 - Troca de índices do vetor de posições.

```
% permute pos_top points so that legs are end points of base and top points
pos_top = [pos_top(6,:); pos_top(1:5,:)]; %6th point on top connects to 1st on bottom
```

Fonte: código “.m” do simulador.

Até este momento, definiu-se todos os pontos em que os atuadores estão acoplados, o que significa dizer que foram configurados os pontos em que as juntas universais estão posicionadas no modelo.

Em seguida, definiu-se a posição dos pontos em que os atuadores estão

acoplados à plataforma superior com relação ao sistema de coordenadas móvel, posicionado no centro de massa da plataforma. As coordenadas X e Y serão as mesmas, afinal, assume-se que os dois sistemas de coordenadas estão inicialmente alinhados, havendo apenas uma translação na direção Z que os diferencie. Esta translação refere-se à altura inicial, salva em “*height*”. Então, para definir os pontos neste novo sistema de coordenadas, basta subtrair o valor da coordenada Z dos valores obtidos anteriormente para o sistema de coordenadas inercial. Isto é justamente o que é feito na linha de código mostrada na Figura 34, dando origem a uma matriz 3x6 (chamada “*body_pts*”), contendo as coordenadas X, Y e Z dos pontos de acoplamento dos seis atuadores à plataforma superior, com relação ao sistema de coordenadas móvel.

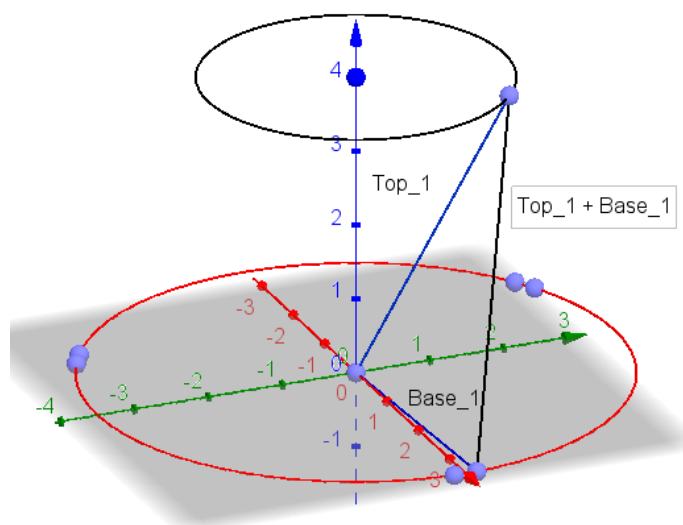
Figura 34 - Pontos com relação ao sistema de coordenadas móvel.

```
% Compute points w.r.t. to the body frame in a 3x6 matrix
body_pts = pos_top' - height*[zeros(2,6);ones(1,6)];
```

Fonte: código “.m” do simulador.

Os pontos de acoplamento nas plataformas podem ser descritos por vetores. Estes vetores têm origem em (0,0,0) e destino nos próprios pontos de acoplamento, identificados no exemplo da Figura 35 como “*Top_1*” para a extremidade do atuador acoplada à plataforma superior, e “*Base_1*” para a extremidade do atuador acoplada à plataforma inferior. Somando-se estes vetores obtém-se um novo vetor com a magnitude e a direção do atuador. Esta soma vetorial pode ser vista com mais clareza na Figura 35, em que o vetor “*Top_1 + Base_1*” representa o resultado para um dos atuadores.

Figura 35 - Soma vetorial.



Fonte: elaborado pelo autor.

Sendo assim, a magnitude do vetor e, portanto, o comprimento dos atuadores na condição inicial, são dados pelo módulo deste vetor resultante. O mesmo acontece como a sua orientação, ou seja, seus cossenos diretores são dados pela divisão das coordenadas X, Y e Z pelo módulo do vetor. Estas operações foram implementadas em código, conforme visto na Figura 36.

Figura 36 - Operações com vetores na direção dos atuadores.

```
% leg vectors
legs = pos_top - pos_base;
leg_length = [ ];
leg_vectors = [ ];
for i = 1:6,
    leg_length(i) = norm(legs(i,:));
    leg_vectors(i,:) = legs(i,:) / leg_length(i);
end
```

Fonte: código “.m” do simulador.

Definindo-se o comprimento inicial dos atuadores, tem-se todo o necessário para que, dado um movimento desejado, seja possível calcular a cinemática inversa do mecanismo.

Os próximos passos são relacionados a configurações mais específicas, como por exemplo, os eixos de rotação das juntas universais, os limites de translação e direção de atuação da junta cilíndrica e as propriedades de massa, dado os materiais e as dimensões dos componentes. A determinação da direção de atuação de cada junta é feita a partir do trecho de código visto na Figura 37.

Figura 37 - Direção de atuação das juntas.

```
% Calculate revolute and cylindrical axes
for i = 1:6,
    rev1(i,:) = cross(leg_vectors(i,:), z_axis);
    rev1(i,:) = rev1(i,:) / norm(rev1(i,:));
    rev2(i,:) = - cross(rev1(i,:), leg_vectors(i,:));
    rev2(i,:) = rev2(i,:) / norm(rev2(i,:));
    cyl1(i,:) = leg_vectors(i,:);
    rev3(i,:) = rev1(i,:);
    rev4(i,:) = rev2(i,:);
end
```

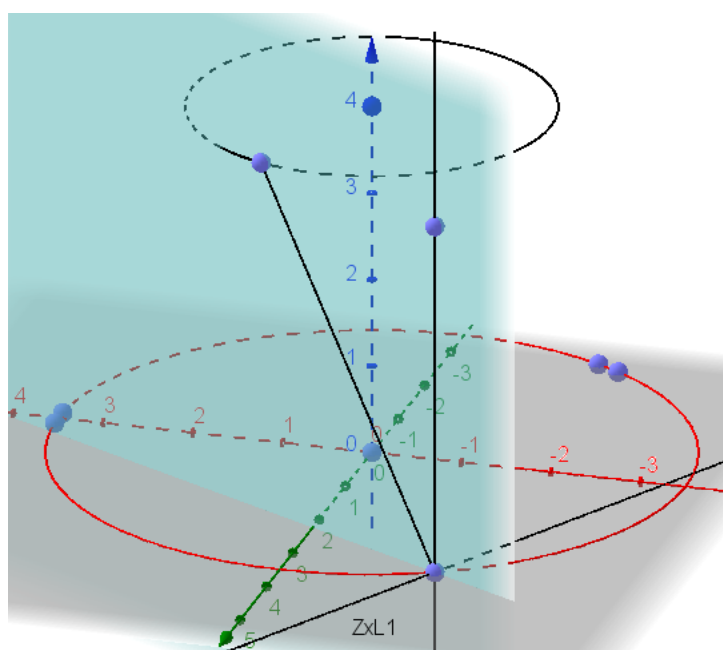
Fonte: código “.m” do simulador.

O mesmo procedimento é feito para todos os pares de juntas universais conectadas a cada um dos 6 atuadores, sendo que “rev1” e “rev2” são os eixos de rotação dos dois graus de liberdade de uma das juntas universais conectada ao atuador, e “rev3” e “rev4” referem-se aos eixos de rotação da outra junta universal conectada a este mesmo atuador. Como as rotações das juntas são complementares, para que haja movimentação, as juntas devem oferecer graus de liberdade em torno dos mesmos eixos, e por isso “rev3” e “rev4” são iguais a “rev1” e “rev2”.

Levando em consideração a ilustração apresentada na Figura 38, a direção do eixo de rotação “rev1” é calculada a partir do produto vetorial entre o vetor do atuador e o eixo Z. Como trata-se de um produto vetorial em que o resultado de interesse é a

direção, para representar o eixo Z traçou-se uma reta paralela a ele (com a mesma direção), como forma de facilitar a compreensão da operação. O eixo paralelo a Z e o vetor do atuador formam um plano, apresentado em azul. Por definição, o produto vetorial entre dois vetores dá origem a um vetor perpendicular ao plano gerado por eles. Desta forma, este produto vetorial resultará em um vetor na mesma direção da reta “ZxL1”, apresentada na Figura 38. Calculando-se o módulo deste vetor resultante salvo em “rev1”, e dividindo suas coordenadas X, Y e Z pelo seu módulo, encontra-se os cossenos diretores deste, ou seja, a orientação deste eixo. Este é um dos eixos em torno do qual, para esta condição inicial, a junta universal acoplada a plataforma inferior rotacionará.

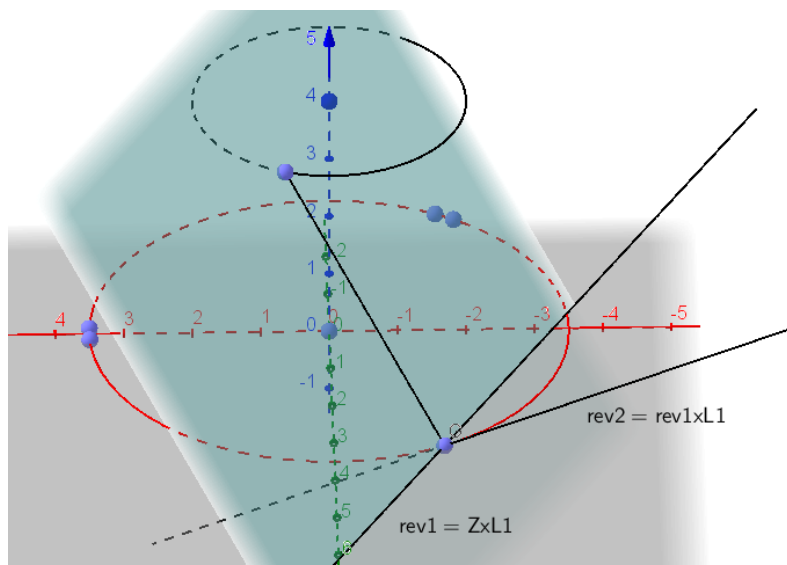
Figura 38 - Ilustração da direção da junta universal 1.



Fonte: elaborado pelo autor.

De forma muito semelhante, pode-se determinar o segundo eixo de rotação da junta universal. Para este segundo eixo, o plano gerado é entre os vetores “rev1” e o vetor do atuador, apresentado em azul na Figura 39. O vetor resultante deste produto vetorial, perpendicular ao plano, é apresentado como “rev2” na Figura 39. Mais uma vez, calculando-se o módulo e dividindo suas coordenadas por este módulo, determina-se a direção deste vetor, que representará o outro eixo de rotação em torno do qual a junta universal estará rotacionando nesta condição inicial.

Figura 39 - Ilustração da direção da junta universal 2.



Fonte: elaborado pelo autor.

Quanto à junta cilíndrica, ela une as partes superior e inferior do atuador linear e só pode se movimentar na direção do próprio atuador, de forma que sua direção de atuação é dada pela orientação “*leg_vector*”.

Uma vez que, conforme descrito no modelo, os atuadores são compostos por dois corpos cilíndricos, deve-se definir suas condições iniciais, como por exemplo: seus pontos de origem, seus pontos máximos e seus eixos de rotação e translação. Para tal, declara-se uma *struct*, como a da Figura 40, com as características descritas.

Figura 40 - Vetores dos componentes do atuador.

```
% Coordinate systems
lower_leg = struct('origin', [0 0 0], 'rotation', eye(3), 'end_point', [0 0 0]);
upper_leg = struct('origin', [0 0 0], 'rotation', eye(3), 'end_point', [0 0 0]);
```

Fonte: código “.m” do simulador.

Neste modelo, definiu-se que para cada atuador o cilindro inferior terá uma de suas extremidades acoplada, seu centro de massa estará originalmente posicionado a três oitavos do comprimento total do atuador, e sua extremidade livre estará a três quartos do comprimento total do atuador. Este componente estará sujeito às rotações e translação determinadas, nas direções calculadas para as juntas universais e para a junta cilíndrica. O cilindro superior, por outro lado, terá uma de suas extremidades acoplada à plataforma superior, enquanto o seu centro de massa estará originalmente a cinco oitavos do comprimento total do atuador. Sua outra extremidade estará localizada a um quarto do comprimento total do atuador. Suas rotações e translação são nas mesmas direções das do cilindro inferior. O código para configurar a *struct* é conforme segue na Figura 41.

Figura 41 - Centro de massa e ponto terminal de componentes do atuador.

```

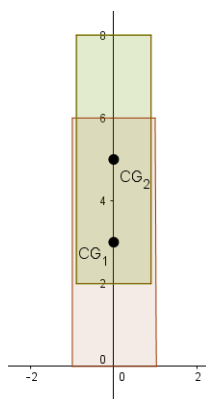
for i = 1:6,
    lower_leg(i).origin = pos_base(i,:) + (3/8)*legs(i,:);
    lower_leg(i).end_point = pos_base(i,:) + (3/4)*legs(i,:);
    lower_leg(i).rotation = [rev1(i,:) ', rev2(i,:) ', cyll1(i,:) '];
    upper_leg(i).origin = pos_base(i,:) + (1-3/8)*legs(i,:);
    upper_leg(i).end_point = pos_base(i,:) + (1/4)*legs(i,:);
    upper_leg(i).rotation = [rev1(i,:) ', rev2(i,:) ', cyll1(i,:) '];
end

```

Fonte: código “.m” do simulador.

Desta forma, a posição de cada um dos corpos que formam o atuador está completamente definida. A Figura 42 pode facilitar a compreensão destas condições definidas. Na Figura 42, a escala vai de 0 a 8 para facilitar a compreensão. Em vermelho tem-se o cilindro inferior, com extremidade inferior fixa em 0, extremidade superior em 6 (ou seja, seis oitavos do comprimento total do atuador), e centro de massa CG_1 em 3 (ou seja, três oitavos do comprimento total do atuador). Em verde, tem-se o cilindro superior, com uma de suas extremidades acoplada em 8, seu centro de massa CG_2 em 5 (ou seja, a cinco oitavos do comprimento total do atuador), sua outra extremidade em 4, a um quarto do comprimento total do atuador.

Figura 42 - Ilustração do atuador e seus componentes.



Fonte: elaborada pelo autor.

Uma vez que todos os parâmetros que definem geometricamente o modelo estão determinados, resta determinar as propriedades de inércia dos elementos. Para tal, é necessário primeiramente definir a espessura de cada um dos elementos. A espessura das plataformas superior e inferior e dos cilindros interno e externo que formam o atuador, são definidas conforme apresentado na Figura 43. Além destes, também é definida a densidade do material do qual os elementos são constituídos. Considerou-se este modelo completamente construído em aço.

Figura 43 - Espessura dos elementos da plataforma.

```
% Inertia and mass calculation
top_thickness = 0.02;
base_thickness = 0.015;
inner_radius = 0.05;
outer_radius = 0.07;
density = 76e3/9.81; % steel density in Kg/m^3
```

Fonte: código “.m” do simulador.

As propriedades de massa e inércia para elementos cilíndricos podem ser obtidas no MATLAB, conforme Figura 44.

Figura 44 - Propriedades de massa e inércia dos atuadores.

```
%leg inertia and mass
[lower_leg_mass, lower_leg_inertia] = inertiaCylinder(density, ...
    0.75*leg_length(1), outer_radius, inner_radius);
[upper_leg_mass, upper_leg_inertia] = inertiaCylinder(density, ...
    0.75*leg_length(1), inner_radius, 0);
```

Fonte: código “.m” do simulador.

Nota-se que os parâmetros da função levam em consideração a densidade, o comprimento e os raios interno e externo. O comprimento de cada cilindro é de 75% do comprimento do atuador. Quanto aos raios internos e externos, nota-se que o cilindro inferior é vazado e o cilindro superior é maciço, conforme já dito anteriormente. O raio externo do cilindro superior coincide com o raio interno do cilindro inferior.

De forma similar, define-se as propriedades de massa e inércia para as plataformas superior e inferior. Porém, conforme especificado nos requisitos do modelo, além do próprio peso da plataforma superior, há ainda uma massa de 20kg sobre ela. O código é como o que segue na Figura 45.

Figura 45 - Propriedades de massa e inércia das plataformas.

```
% top and base plate mass and inertia
[top_mass, top_inertia] = inertiaCylinder(density, ...
    top_thickness, radius_t, 0);
top_mass = top_mass + 20;
[base_mass, base_inertia] = inertiaCylinder(density, ...
    base_thickness, radius_b, 0);
```

Fonte: código “.m” do simulador.

Por fim, deve-se determinar os coeficientes proporcional, derivativo e integrativo do controlador PID responsável por controlar a Plataforma de Stewart. Os coeficientes capazes de cumprir com os requisitos do projeto são mostrados na Figura 46.

Figura 46 - Coeficientes do controlador PID.

```
% PID controller gains
Kp = 2e4; Ki = 1e2; Kd = 4.5e2;
```

Fonte: código “.m” do simulador.

Para facilitar a visualização das características que definem a plataforma, os dados foram agrupados na Tabela 2.

Tabela 2 – Dados da plataforma de Stewart do modelo

Altura na posição de partida	0,4 metros
Raio da plataforma inferior	0,35 metros
Raio da plataforma superior	0,2 metros
Espessura da plataforma superior	0,02 metros
Espessura da plataforma inferior	0,015 metros
Raio interno da camisa do pistão	0,05 metros
Raio externo da camisa do pistão	0,07 metros
Raio da haste maciça do pistão	0,05 metros
Coeficiente Proporcional do PID (Kp)	20000
Coeficiente Integrativo do PID (Ki)	100
Coeficiente Derivativo do PID (Kd)	4500

5.1.4 RESULTADOS DO MODELO

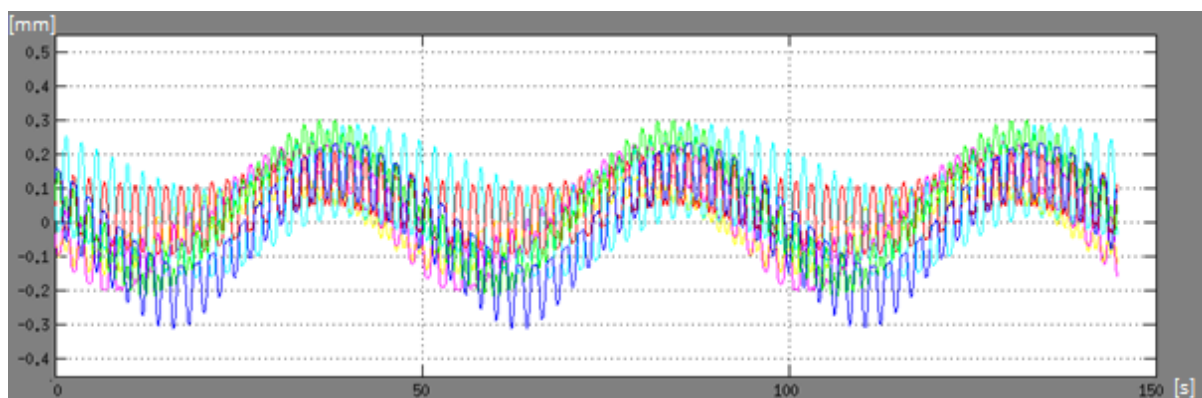
O modelo tem como pré-requisitos um intervalo de movimentação de 300 mm nas três direções, 60 graus de angulação em torno dos 3 eixos, e capacidade de carga de 20 kg. Além disso, deverá ter uma velocidade máxima de 20mm/s.

A carga, conforme visto na Figura 45, já foi considerada no modelo. No que diz respeito ao intervalo de movimentação, seja de translação ou rotação, deve-se determinar o curso necessário aos atuadores para atendê-lo. Já no que concerne à velocidade, deve-se estabelecer um movimento desejado, uma vez que o controlador PID não é adaptativo, ou seja, para um determinado movimento o controlador é eficiente, enquanto pra outro pode não ser. Sendo assim, o controlador deve ter seus

coeficientes determinados para o movimento desejado. Para o desenvolvimento deste projeto, analisou-se dois tipos de movimentação, ambas de translação no eixo Z. A primeira, que trata-se de um movimento senoidal, busca estabelecer qual a capacidade de carga necessária a cada atuador para atingir, para este movimento, a velocidade desejada para a plataforma superior, levando em conta uma carga de 20kg. A segunda, trata-se de um conjunto de degraus e tem como objetivo mostrar a influência dos parâmetros do PID no tempo de resposta da plataforma, bem como a influência deste tempo de resposta na capacidade de carga necessária aos atuadores.

Analisando-se, em um primeiro momento, os intervalos de movimentação, a Figura 47 mostra o delta de comprimento para todos os atuadores, quando estes executam movimentos senoidais de mesma frequência, tanto para translação quanto para rotação. As senóides de posição foram definidas com amplitude de 300mm, variando entre 150 e -150mm em todos os eixos. As senóides do movimento angular tem amplitude de 60 graus, variando entre 30 e -30 graus em torno de todos os eixos.

Figura 47 - Delta de comprimento dos atuadores.

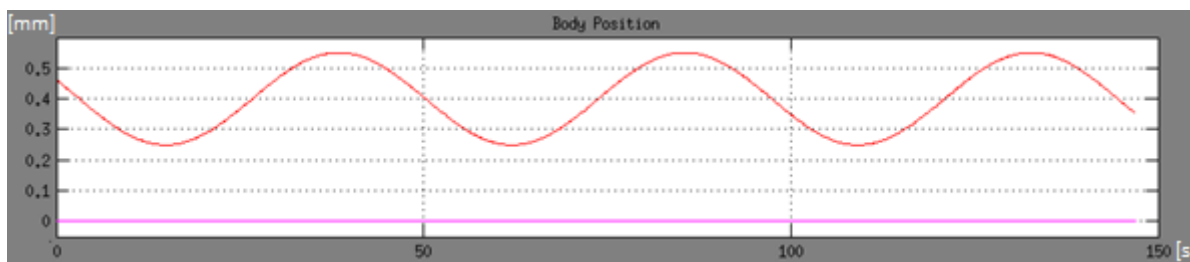


Fonte: elaborada pelo autor.

Como a frequência é a mesma, os pontos de máximo em cada movimento são atingidos ao mesmo tempo, gerando o máximo de expansão para parte dos atuadores, e o máximo de contração para outros. Sendo assim, com um único movimento pode-se determinar qual o curso máximo necessário aos atuadores. Nota-se que as maiores variações estão na casa dos 300mm, positivo e negativo. Sendo assim, para o amplitude de movimentação requerido, necessita-se de atuadores com curso de 600mm.

Para analisar a capacidade de carga de cada um dos atuadores, aplicou-se um movimento senoidal na direção Z de amplitude 300mm e frequência 0,02 dividido por 0,15, de forma que a velocidade resultante seja uma cossenóide de amplitude 20mm/s, especificado como requisito para o modelo simulado. O movimento gerado para a plataforma superior equivale ao visto na Figura 48.

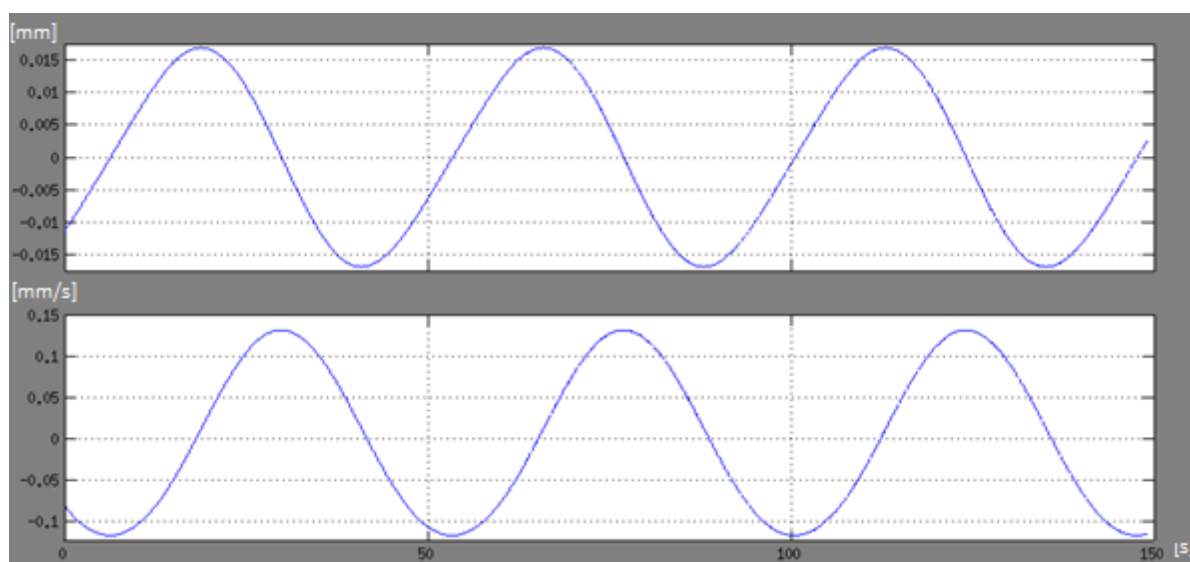
Figura 48 - Movimento da plataforma superior.



Fonte: elaborada pelo autor.

Como resultado do movimento aplicado, a velocidade e o deslocamento para cada atuador é equivalente ao visto na Figura 49.

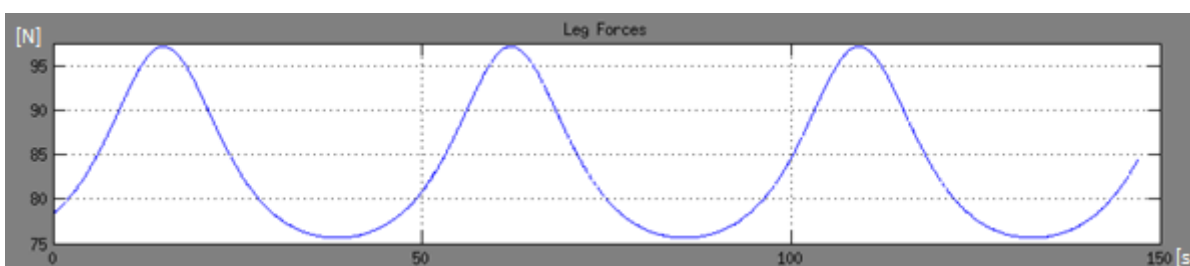
Figura 49 - Velocidade e deslocamento para movimento aplicado.



Fonte: elaborada pelo autor.

Para estas condições aplicadas ao modelo, segundo os requisitos estabelecidos para o mesmo, as forças necessárias aos atuadores são como mostra a Figura 50.

Figura 50 - Forças exercidas pelos atuadores.



Fonte: elaborada pelo autor.

O que significa dizer que, para este movimento senoidal na direção Z,

atendendo os requisitos de velocidade máxima e de amplitude do deslocamento, a capacidade de carga para cada atuador, considerando-se uma carga 20kg, deve ser no mínimo de 100N.

Vale ressaltar mais uma vez a vantagem decorrente do uso de mecanismos paralelos no que diz respeito a sua maior capacidade de carga. Enquanto que estaticamente a carga de 20kg já exerce uma força de 200N, dinamicamente esta força é ainda superior e, ainda assim, a capacidade de carga necessária a cada atuador nem se quer é superior a carga estática do corpo sobre a plataforma.

Analizando os erros entre a posição desejada e a lida durante o movimento na Figura 51, nota-se que se trata de uma curva suave, o que quer dizer que o movimento é suave e não exige que a força, por parte dos atuadores, seja de grande magnitude. Estes erros darão origem ao sinal de *feedback*, no qual o PID se baseará para determinar a força de atuação.

Figura 51 - Sinal de entrada do controlador.

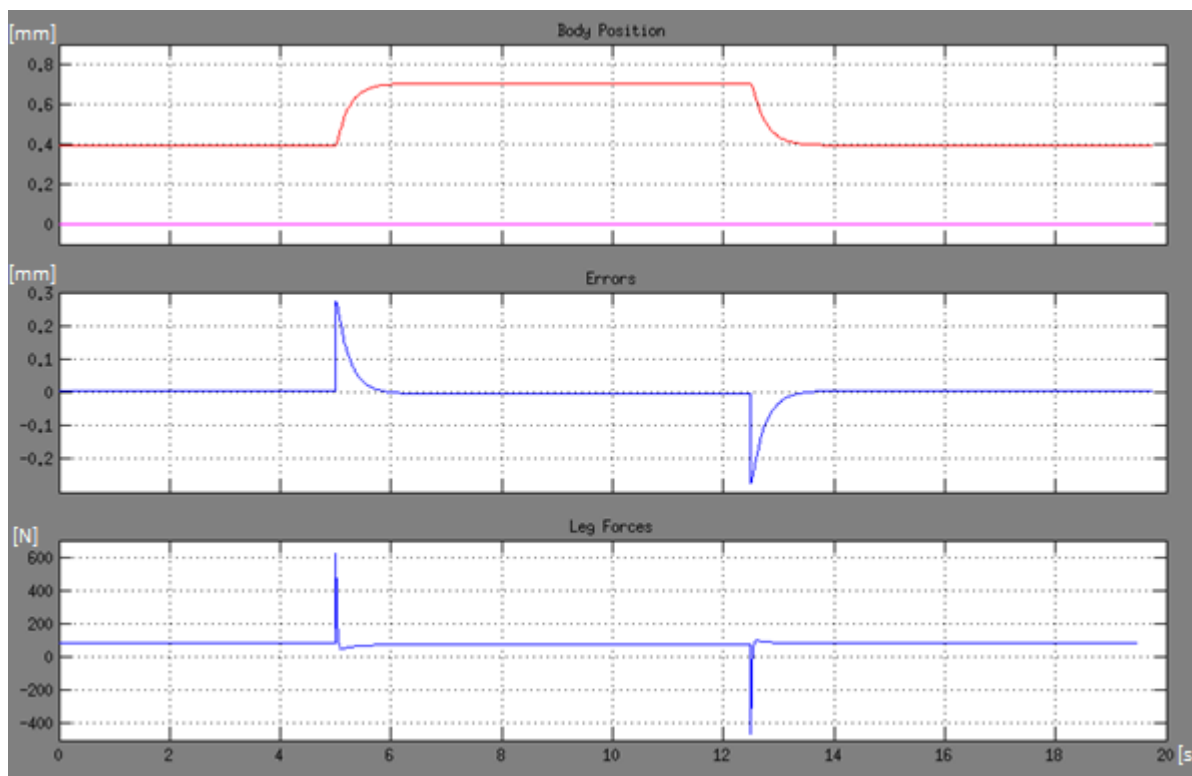


Fonte: elaborada pelo autor.

Porém, como já foi dito, para cada tipo de movimento o modelo se comportará de uma forma diferente e, como o modelo é governado pelo PID, os parâmetros deste também influenciam fortemente a capacidade de carga de cada atuador.

A fim de exemplificar, na Figura 52 encontram-se as curvas de posição da plataforma superior, erro entre posição desejada e posição atual e, por fim, a força necessária aos atuadores para executar o movimento. O movimento é na mesma direção e trata-se de um degrau de amplitude 300mm. Embora esteja fora dos requisitos determinados ao projeto, servirá bem como parâmetro para a análise aqui feita.

Figura 52 - Resposta suave ao degrau.

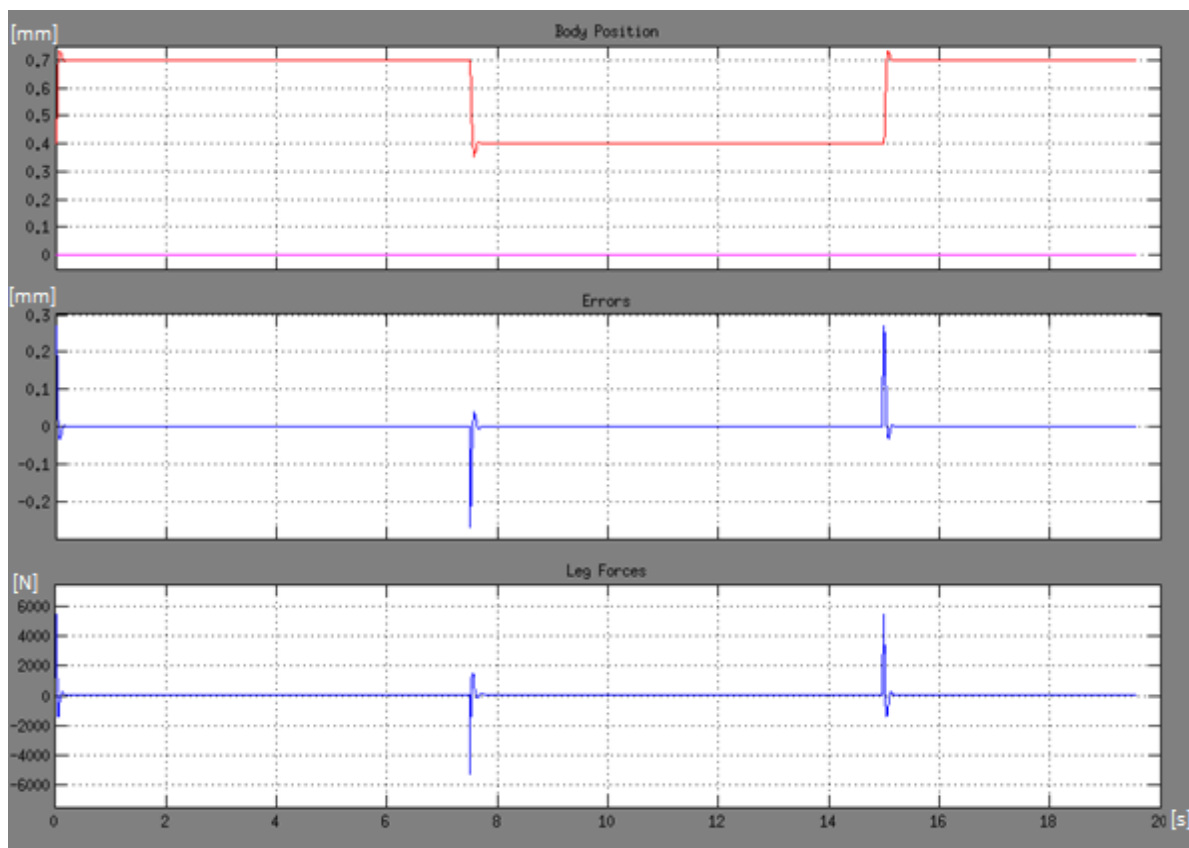


Fonte: elaborada pelo autor.

Para esta movimentação, tratando-se de um degrau, ou seja, uma variação brusca, as velocidades precisam atingir valores mais elevados para reduzir o erro e atingir a posição desejada no intervalo de tempo desejado e, conseqüentemente, a força exercida por cada atuador é maior, na casa dos 500 N. Mas a análise aqui refere-se ao ajuste do controlador. Neste caso, o controlador tem um tempo de acomodação de meio segundo, ou seja, uma vez que o degrau seja aplicado, o controlador leva meio segundo para deslocar a plataforma superior à posição desejada.

Na Figura 53 é possível observar a resposta do sistema a um controlador com o tempo de acomodação mais curto. Similarmente à Figura 52, esta também apresenta as curvas de posição da plataforma superior, erro entre posição desejada e posição atual e, por fim, a força necessária aos atuadores.

Figura 53 - Resposta abrupta ao degrau.



Fonte: elaborada pelo autor.

No caso deste segundo controlador, o tempo de resposta é expressivamente menor, ou seja, em 120 ms após a aplicação do degrau, o PID já faz com que a plataforma superior atinja a posição desejada. Como consequência, a força necessária a cada atuador vai a aproximadamente 5000 N. Para uma plataforma hidráulica, este tempo de resposta não é viável, mas o objetivo aqui é ilustrar a influência do movimento na determinação da força necessária ao mecanismo.

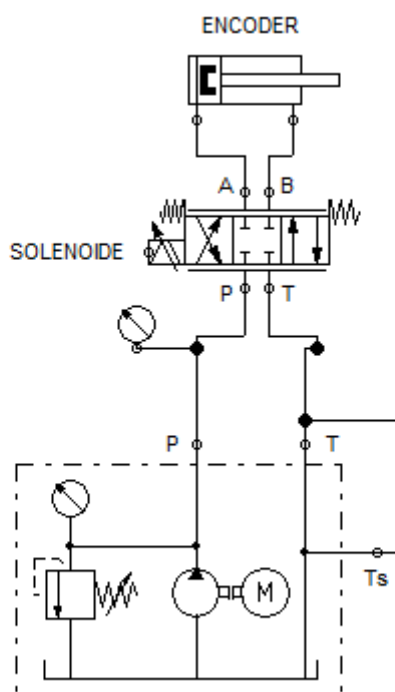
Como resultado desta análise, observa-se que, para determinar a capacidade de carga necessária a cada atuador, seria necessário requisitos específicos de movimentações e velocidades, bem como o tempo de resposta desejado para o sistema, uma vez que a capacidade de carga necessária pode variar para cada movimento. Dentro dos requisitos de amplitude na movimentação, há infinitas possibilidades de movimentos diferentes, portanto, infinitas curvas de carga para cada atuador, tornando inviável o dimensionamento preciso de cada atuador. Além disso, para cada movimento desejado, novos coeficientes devem ser determinados para o controlador, uma vez que este não é adaptativo, ou seja, não se adequa a diferentes condições. Isto significa dizer que, para aplicações reais em que vários tipos de movimentos são necessários, deverão haver vários conjuntos de coeficientes para o controlador, sendo que cada conjunto diz respeito a um determinado movimento. Dependendo da quantidade de movimentos, pode vir a ser inviável o uso deste tipo de controlador, sendo mais adequado o uso de controladores mais complexos, porém,

adaptativos.

5.1.5 REPRESENTAÇÃO EM FLUIDSIM

Para que fique mais claro a confecção de uma Plataforma de Stewart com atuadores hidráulicos, foi elaborada uma representação dos circuitos hidráulico e elétrico da plataforma com o uso do *software* FluidSim. As Figuras 54 e 55 ilustram esta representação.

Figura 54 - Representação do circuito hidráulico.

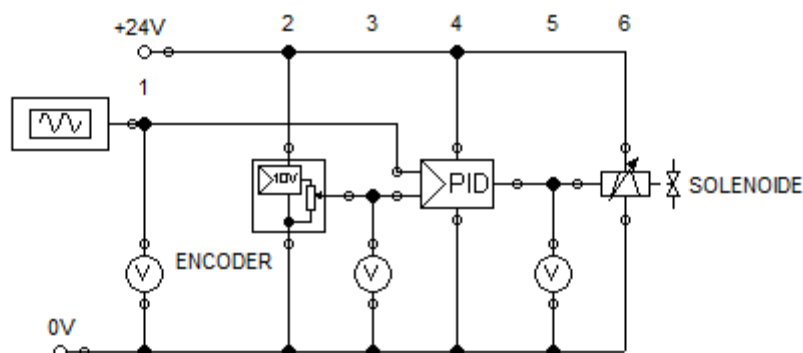


Fonte: elaborada pelo autor.

Na Figura 54 é representado o circuito hidráulico de um dos seis atuadores, contando com uma unidade hidráulica responsável por fornecer a energia hidráulica necessária ao movimento, uma válvula proporcional pilotada por solenoide, um manômetro e o atuador hidráulico com um sensor de deslocamento. Imaginando que, conforme resultado encontrado anteriormente, cada cilindro de área $0,007458\text{m}^2$ necessite exercer forças que cheguem a 100N , a pressão necessária a esta unidade hidráulica será de aproximadamente $12,5\text{KPa}$ manométrica. Já que a velocidade máxima é de 200 mm/s , ou seja $0,2\text{m/s}$, a vazão necessária a cada atuador será o produto de sua área pela velocidade de seu deslocamento, ou seja, $0,014916\text{ m}^3/\text{s}$. A bomba deve ser capaz de suprir esta vazão aos seis atuadores, ou seja, a bomba

deve ter vazão volumétrica mínima de 0,0089496 m³/s.

Figura 55 - Representação do circuito elétrico.



Fonte: elaborada pelo autor.

A representação do circuito elétrico, conforme a Figura 55, contém um gerador de sinais senoidais, um receptor do sinal do sensor de deslocamento, um controlador PID e a pilotagem do solenoide. O gerador de sinais representa, neste exemplo, o *software* de geração de movimentos desejados que é entrada do PID. Além destes, também há alguns volímetros para monitoramento de tensão e uma fonte de tensão constante de 24 V.

Em uma plataforma real, um software geraria o movimento, receberia o sinal digital do sensor de posição, os utilizaria como sinais de entrada em um PID digital e emitiria um sinal ao solenoide, que pilotaria a válvula no circuito hidráulico.

5.2 PROTÓTIPO SIMPLIFICADO

Com o objetivo de exemplificar os seis graus de liberdade da Plataforma de Stewart, foi proposto o desenvolvimento de um protótipo simplificado. Pode ser considerado simplificado porque os componentes utilizados não garantem boa precisão, há diferença entre as juntas utilizadas e as consideradas durante o desenvolvimento dos modelos cinemáticos implementados e os modelos cinemáticos foram adaptados para o uso do mecanismo biela-manivela. Os modelos cinemáticos estudados levam em consideração a utilização de duas juntas esféricas, enquanto o modelo utiliza duas juntas universais. Foi considerado que, apesar das condições citadas, o protótipo simplificado ainda serviria ao seu propósito. É importante deixar claro que o desenvolvimento deste protótipo não tem relação com o modelo computacional apresentado nos tópicos anteriores. Os modelos cinemáticos utilizados neste protótipo simplificado são diferentes do utilizado no modelo computacional. As características construtivas também não têm relação.

Para o desenvolvimento do protótipo, alguns modelos cinemáticos inversos e

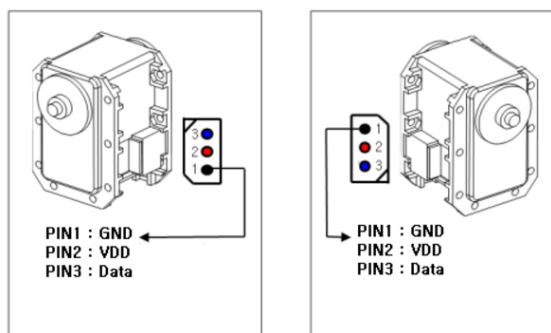
diretos foram estudados, como o apresentado por Nanua, Waldron e Murthy (1990), que apresenta como solução para o modelo cinemático direto uma equação de 16ª ordem com coeficientes pares. Os autores Nanua, Waldron e Murthy (1990) citam que a solução analítica do problema da cinemática direta resulta em uma equação de 64ª ordem, porém, com o uso do método de Berzout, conseguiram alcançar a solução de 16ª ordem contendo apenas coeficientes pares. Após o entendimento do modelo, as equações foram implementadas em MATLAB, porém o tempo de processamento tornou a implementação inviável para o protótipo. Na sequência, foi feito o estudo de um modelo cinemático direto com resolução numérica, apresentado por Liu, Lewis e Fitzgerald (1994). O modelo tira vantagem das formas geométricas das plataformas superior e inferior, que para sua implementação, precisam possuir forma definida. O equacionamento dos modelos cinemáticos direto e inverso desenvolvidos por Liu, Lewis e Fitzgerald (1994) podem ser vistos no Apêndice A.

Nesta resolução, os autores chegam a 3 equações e 3 variáveis como resultado para o problema da cinemática direta. Porém, as equações são não lineares e, para resolução, os autores sugerem o uso do método numérico de Newton-Raphson. A implementação do método numérico e seus resultados são apresentados também no Apêndice A. Estes modelos também foram implementados em MATLAB para avaliar o custo computacional, validar a formulação e realizar simulações. O código implementado em MATLAB pode ser encontrado no Apêndice 4.

Uma vez determinados os modelos cinemáticos direto e inversos que seriam utilizados para o desenvolvimento do protótipo simplificado da Plataforma de Stewart, iniciou-se a construção do protótipo. Cada componente para construção do protótipo simplificado pode ser encontrado no Apêndice 3.

Foram utilizados motores como atuadores e para controlá-los, a fim de realizar os movimentos gerados em computador, utilizou-se um driver de conversão do padrão de tensão TTL (*Transistor-Transistor Logic*) para USB (*Universal Serial Bus*), ou seja, um driver que converte o padrão de tensão TTL utilizado pelos motores para o padrão de tensão USB utilizado pelo computador. Para alimentá-los, utilizou-se uma bateria de LIPO de 3 células de 11.1V de tensão nominal. Foram utilizados cabos de 3 vias para conectar os motores ao driver e à alimentação, um cabo de 2 vias para transformar o padrão do conector da bateria no padrão de conexão utilizado nos motores e um multiplicador de conectores, também do padrão utilizado pelos motores. O padrão de conexão utilizado pelos motores é como o apresentado na Figura 56.

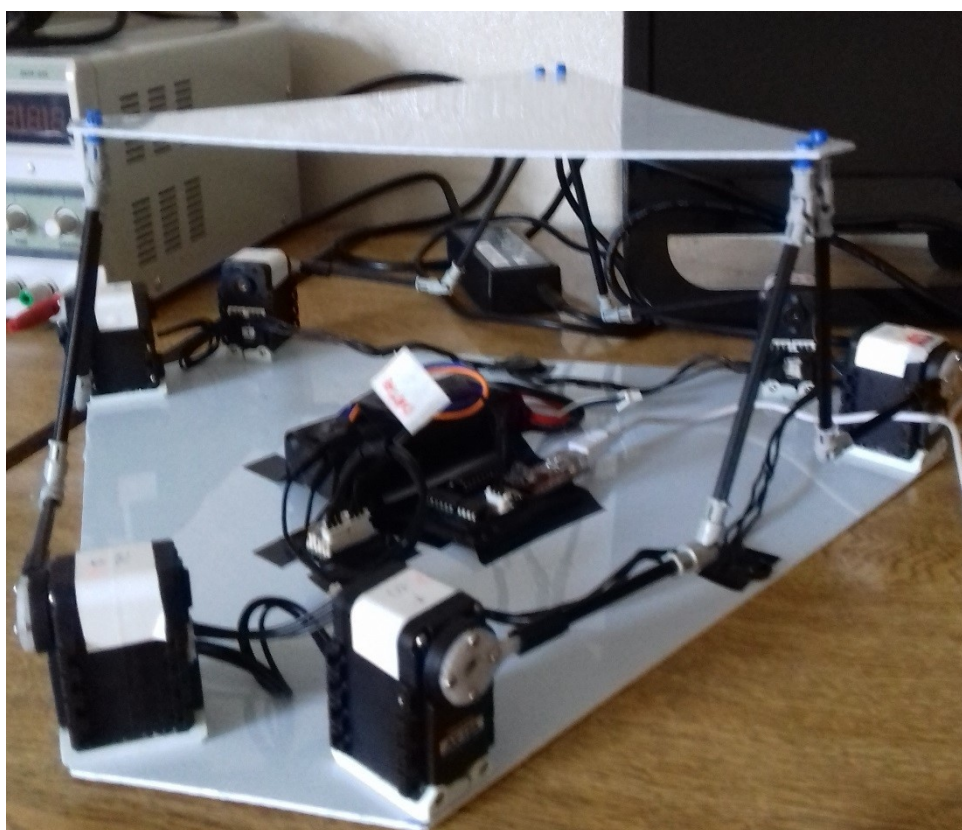
Figura 56 - Padrão TTL dos motores.



Fonte: site robot-italy.com¹²

Utilizando os componentes listados no Apêndice 3, a plataforma foi construída e o resultado final pode ser visto na Figura 57.

Figura 57 - Protótipo construído.

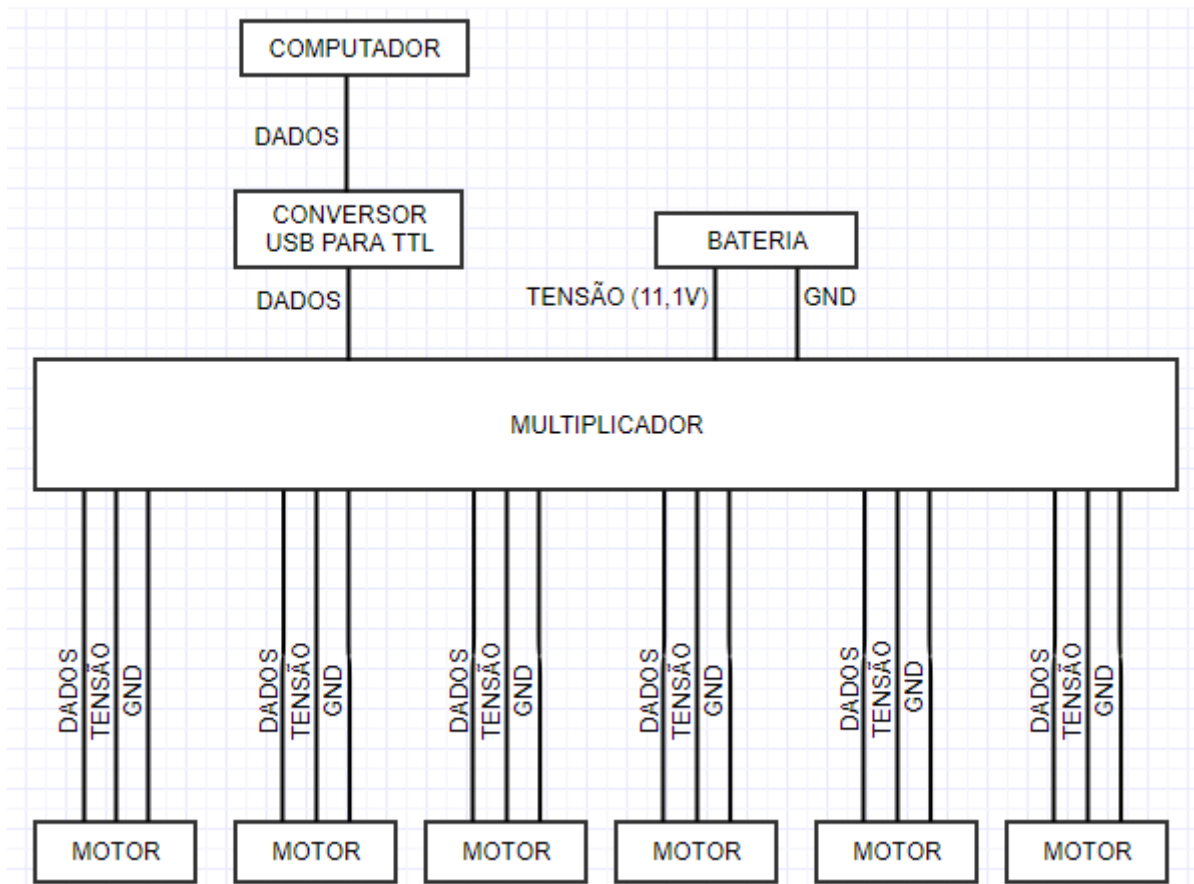


Fonte: elaborado pelo autor.

O diagrama da Figura 58 ajuda na compreensão do funcionamento do protótipo.

¹² Figura 56 disponível em: <https://www.robot-italy.com/en/dynamixel-mx28-robot-servo.html>, acesso 26/07/2017

Figura 58 - Diagrama do protótipo.



Fonte: elaborado pelo autor.

O computador gera a trajetória e envia os dados via USB ao driver que converte os dados para TTL. A bateria fornece a tensão de 11.1V, fechando o circuito de potência. Alimentação e dados são transmitidos a todos os motores a partir do multiplicador.

5.2.1 MOTORES

Os motores AX-12, similares ao visto na Figura 59, executam ações baseadas em comandos que são interpretados pelo motor de acordo com um protocolo definido. Segundo os exemplos de uso no Anexo A e o *datasheet* do motor no Anexo B, o motor executa aquilo que está salvo em sua memória, podendo executar imediatamente ou aguardar um comando de ação para fazê-lo. O que vai determinar a forma a ser executada é a instrução passada ao motor.

Figura 59 - AX-12A.



Fonte: site support.robotis.com¹³

Todos os motores conectados recebem todos os pacotes de dados enviados a nível de camada de enlace. Quando o motor inicia a interpretação dos dados recebidos, verifica se o pacote é ou não direcionado a ele. Caso os dados sejam destinados a ele, o motor é capaz de identificar a instrução e o parâmetro da mesma, atuando então como especificado pelo pacote de dados.

O pacote de dados, segundo o protocolo, tem partes comuns a todas as instruções e outras que variam conforme a instrução. De forma simplificada, pode-se definir o protocolo da seguinte forma: em seus dois primeiros bytes, há apenas bits 1, sendo essa a maneira que o motor tem de identificar ou notificar que este é um início de um pacote de dados. No terceiro byte, vem o ID que nada mais é que um identificador do atuador. Até este ponto, todos os pacotes são abertos na camada de enlace. A partir da informação de ID contida neste campo, o motor é capaz de determinar se o pacote de dados é direcionado a ele ou não. O quarto byte trata-se do comprimento do restante do pacote, sendo a soma dos bytes na sua sequência até o fim do pacote, os 3 primeiros bytes, e ele mesmo. Por meio deste byte, o motor saberá quantos endereços de memória o pacote pretende modificar. O quinto byte diz respeito à instrução que se deseja passar ao motor, que neste caso, será sempre a instrução “*WRITE*”, que escreve em um endereço da memória do motor e executa a ação recorrente da mudança imediatamente. Em seguida, vêm os parâmetros da instrução, e para a instrução “*WRITE*”, o primeiro sempre diz respeito ao endereço de memória inicial que se deseja alterar. Só é possível alterar endereços de memória sequenciais. Caso seja necessário alterar endereços não contíguos, é preciso enviar dois pacotes. Os bytes em sequência ao endereço de memória inicial representam os novos valores a serem escritos na memória do motor. O último byte é o “*CHECKSUM*”, responsável pela verificação de integridade do pacote. Trata-se de uma operação realizada com os dados de pacote quando montado que, quando recebido pelo motor, se apresentar um resultado inconsistente, sinaliza a inconsistência dos dados.

¹³ Figura 59 disponível em: http://support.robotis.com/en/product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm, acesso 26/07/2017

O conjunto de possíveis instruções podem ser vistas no Anexo A. Mais informações a respeito do motor além das aqui apresentadas, como seus endereços de memória, seus modos de operação, torque máximo, entre outros, podem ser encontrados no manual do fabricante. O manual encontra-se no Anexo B.

5.2.2 COMUNICAÇÃO

Para que o motor execute os movimentos necessários para gerar a movimentação desejada da plataforma superior, é necessário que as informações sejam montadas de acordo com o protocolo do motor e, posteriormente, enviadas via USB. Tais informações precisam ser convertidas para o padrão de tensão TLL, para que, além de estarem formatadas conforme o protocolo do componente, também estejam no padrão de tensão compreendido pelos motores.

Sendo assim, é necessário configurar a comunicação USB e montar o pacote de dados de acordo com o protocolo do motor via software.

Quem faz a interface entre os padrões de tensão é o driver. Sendo assim, para configurar a USB, o primeiro passo é determinar em qual USB o driver foi conectado, e então configurar essa porta USB para se comunicar na mesma velocidade em que o motor está configurado para se comunicar. Conforme visto no Anexo B, para alterar a velocidade de comunicação dos motores, deve-se alterar o campo de memória relativo ao *baudrate*. A relação do *baudrate* com a velocidade de comunicação é dada pela Equação 4.

$$Baudrate = \frac{2000000}{(baudNum+1)} \quad (4)$$

Neste protótipo, utilizou-se o *baudrate* igual a 1, ou seja, uma comunicação 1Mbit/s (1 mega bits por segundo).

Para configurar a velocidade de comunicação da USB, utilizou-se o sistema operacional LINUX UBUNTU 14.04. Para os sistemas operacionais da distribuição LINUX, os dispositivos periféricos, como por exemplo o driver conectado à USB, são abstraídos como arquivos que, quando devidamente configurados, podem ser manipulados com operações realizadas pelo sistema, chamadas *system calls*, de forma que a interação com os dispositivos se resume a operações de leitura e escrita em arquivo, ou seja, escrever no arquivo do driver conectado à USB é equivalente a enviar um pacote de dados via USB, e ler deste arquivo significa ler o pacote de dados enviado pelo motor ao computador. Mas, como dito, a porta precisa ser devidamente configurada. Para tal, faz-se uso de outras *system calls*, capazes de alterar as estruturas (*structs*) responsáveis pelas configurações da porta USB. Configurando estas *structs*, configura-se o modo de operação da porta e a sua velocidade.

Estas *structs* são chamadas “termios” e “serial_struct”. Para configurar a USB,

cria-se *structs* destes formatos e atribui-se a cada um dos elementos da *struct* palavras correspondentes às configurações desejadas. Por meio do comando “tcflush”, os dados da *struct* são apagados e, por meio do comando “tcsetattr”, salva-se novos valores na *struct* do tipo “termios”. Por meio do comando “ioctl”, é possível abrir e configurar a *struct* do tipo “serial_struct”, sendo que a operação é definida pelo parâmetro passado para função, ou seja, se o segundo parâmetro é “TIOCGSERIAL” a função “ioctl” abre a *struct* para configuração, se o segundo parâmetro é “TIOCSSERIAL” a função “ioctl” salva os valores atribuídos nos elementos da *struct*. O código que faz as configurações das *structs* encontra-se no tópico “Dxl_hal.c” do Apêndice 6. Abre-se então, utilizando o comando “open”, o arquivo referente ao dispositivo e ao descritor de arquivo, fornecidos pelo sistema, e então associa-se a este as *structs* configuradas, o que significa configurar a porta como desejado.

Para esta aplicação, a única configuração necessária é a taxa de transmissão de dados, que deve ser configurada em 1Mbit/s. Para isso, na *struct* do tipo “termios”, configura-se o “baud_base”, que virá a ser parte do cálculo da taxa de transmissão. Na *struct* do tipo “serial_struct”, define-se o “custom_divisor”. A velocidade de comunicação da USB é então dada, conforme Equação 5, por:

$$Baudrate_{USB} = \frac{baud_{base}}{custom_{divisor}} \quad (5)$$

Porém, a empresa que produz os motores também produz um driver que faz a interface entre os motores e o computador e, além disso, o software em linguagem C para realizar as configurações de comunicação dos dispositivos, bem como montar os pacotes. O software pode ser encontrado para *download* no *link* da referência [17] com o nome “DXL_SDK_LINUX_v1_01.zip”. O funcionamento do software disponibilizado pode ser visto no *link* da referência [18]. Alguns exemplos de utilização podem ser vistos no *link* da referência [19]. Demais informações a respeito dos dispositivos produzidos pela companhia podem ser encontrados no link da referência [20]. Estes códigos são específicos para comunicação utilizando o driver da empresa, porém, o driver utilizado no desenvolvimento deste protótipo tem funcionamento muito similar, de forma que o mesmo software pôde ser utilizado.

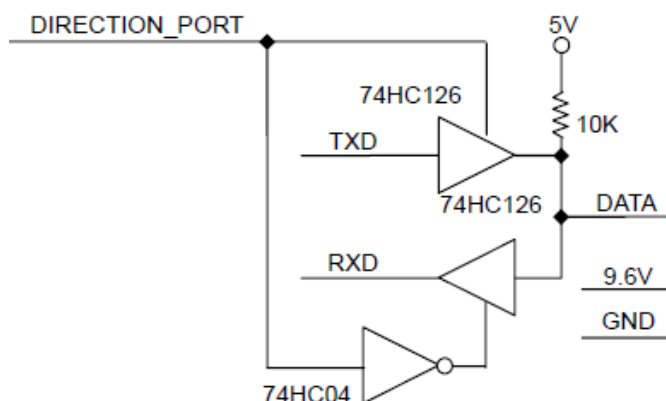
Neste protótipo, o software padrão fornecido pelo fabricante sofreu algumas modificações que serão abordadas posteriormente. Nada foi retirado, apenas acrescido com o objetivo de estruturar melhor o código e torná-lo mais compreensivo. Porém, toda e qualquer atividade aqui realizada pode ser reproduzida utilizando a documentação sugerida.

No que diz respeito ao aspecto construtivo e ao funcionamento do driver, este primeiro realiza a conversão dos dados provenientes da USB para o padrão UART (*Universal Asynchronous Receiver Transmitter*). Para tal, faz-se uso de um circuito integrado chamado FT232-RL. Este circuito integrado é o responsável por transformar os dados provenientes da USB no padrão UART, resultando em dados de escrita e leitura do tipo TX(*Transmitter*)/RX(*Receiver*), necessário à comunicação compatível

com o motor.

Porém, a comunicação UART é *full-duplex*, ou seja, é capaz de fazer escritas e leituras ao mesmo tempo, enquanto a comunicação TTL é *half-duplex*, ou seja, em um mesmo momento, é capaz de fazer só um dos dois, leitura ou escrita. Para transformar a UART *full-duplex* em *half-duplex* usa-se o circuito mostrado na Figura 60.

Figura 60 - Conversão full-duplex para half-duplex.

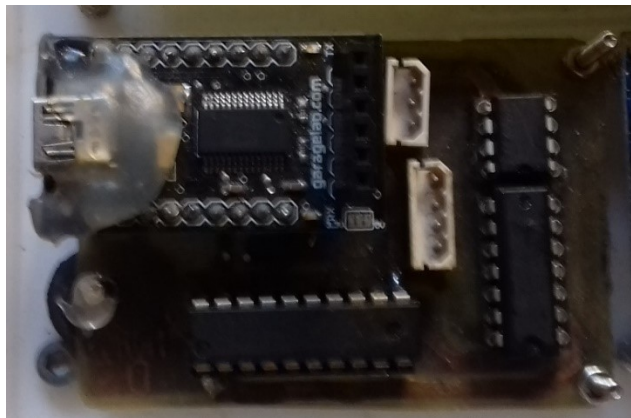


Fonte: site robot-italy.com¹⁴

Os circuitos integrados 74HC126 fazem o papel de *buffers* das portas TXD e RXD, enquanto o circuito integrado 74HC04 é o responsável por habilitar apenas um dos 74HC126 por vez, de modo que DATA, que é o canal de comunicação do conector TTL, seja RX ou TX dependendo da necessidade, ou seja, comute entre leituras e escritas. O *bit* de direção, que é um *bit* emitido pelo circuito eletrônico FT232-RL para informar se a operação na USB é de escrita ou leitura, chamado “DIRECTION_PORT”, é o que determina o funcionamento do circuito integrado 74HC04, e dependendo do valor de tensão em “DIRECTION_PORT”, ou seja, se o nível lógico é alto ou baixo, 0 ou 1, o circuito integrado 74HC04 atuará em um dos circuitos integrados 74HC126 ativando-o e permitindo que o que esteja armazenado no *buffer* de escrita seja enviado, ou o que esteja armazenado no *buffer* de leitura seja lido. Uma vez confeccionado, o driver fica como o da Figura 61.

¹⁴ Figura 60 disponível em: <https://www.robot-italy.com/en/dynamixel-mx28-robot-servo.html>, acesso 26/07/2017

Figura 61 - Driver de comunicação.



Fonte: elaborada pelo autor.

Ao circuito de comunicação é conectado ao terra do circuito de alimentação, de modo a fazer com que todos os componentes tenham um terra comum. Um cabo TTL com os dados é conectado ao circuito de comunicação e a um multiplicador de entradas TTL.

5.2.3 SOFTWARE

A plataforma superior é capaz de realizar qualquer movimento, em função dos seus seis graus de liberdade. Porém, este movimento deve respeitar as limitações de comprimento dos atuadores e as condições de rigidez da plataforma.

Para gerar o movimento desejado de forma a garantir que este não contradiga as condições de rigidez da plataforma, usa-se modelos cinemáticos. Os modelos cinemáticos utilizados neste trabalho tomam como base o conteúdo apresentado nos artigos das referências [10] e [15]. O modelo proposto no artigo da referência [15] apresenta uma solução complexa e de alto custo computacional, o que não se aplicava à proposta de simplificação do modelo. Sendo assim, os modelos matemáticos utilizados para desenvolvimento do protótipo são os apresentados no artigo da referência [10].

Para o desenvolvimento do software que viria a controlar a Plataforma de Stewart, implementou-se os modelos cinemáticos direto e inverso em MATLAB, conforme Apêndice 4, utilizando os métodos apresentados no Apêndices 1 e 2, baseados no artigo da referência [10]. O objetivo foi verificar a validade dos modelos e garantir que os erros, provenientes do cálculo numérico, não fossem de magnitude prejudicial.

Para validar os modelos, um movimento foi gerado e o ângulo de cada um dos atuadores foi calculado, utilizando o modelo cinemático inverso. Para os comprimentos dos atuadores, em cada instante de tempo, retornados como resposta

do modelo cinemático inverso, foi aplicado o modelo cinemático direto e verificado se os valores obtidos para as coordenadas da plataforma superior condiziam com os valores gerados no movimento.

Conforme esperado, o modelo numérico não é capaz de retornar o valor exato fornecido na modelagem, mas os valores obtidos possuem erros muito pequenos, na casa de 10^{-12} cm para posição, e 10^{-14} graus para a orientação, conforme pode ser observado nas Figuras 62 e 63.

Figura 62 - Erro de posição no método numérico.

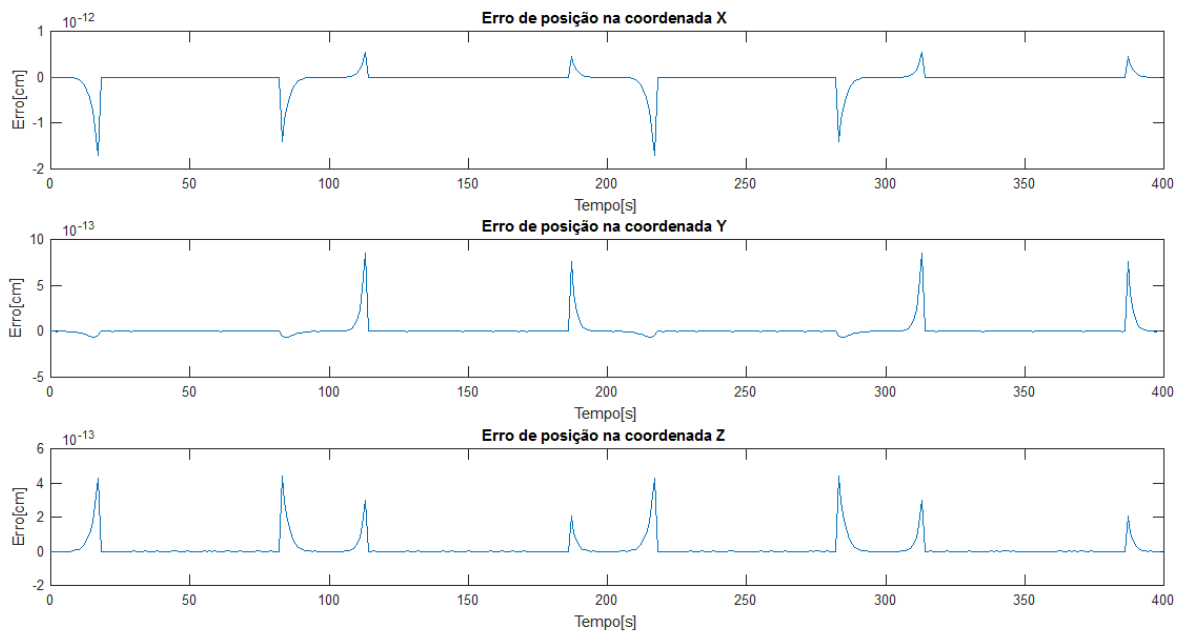
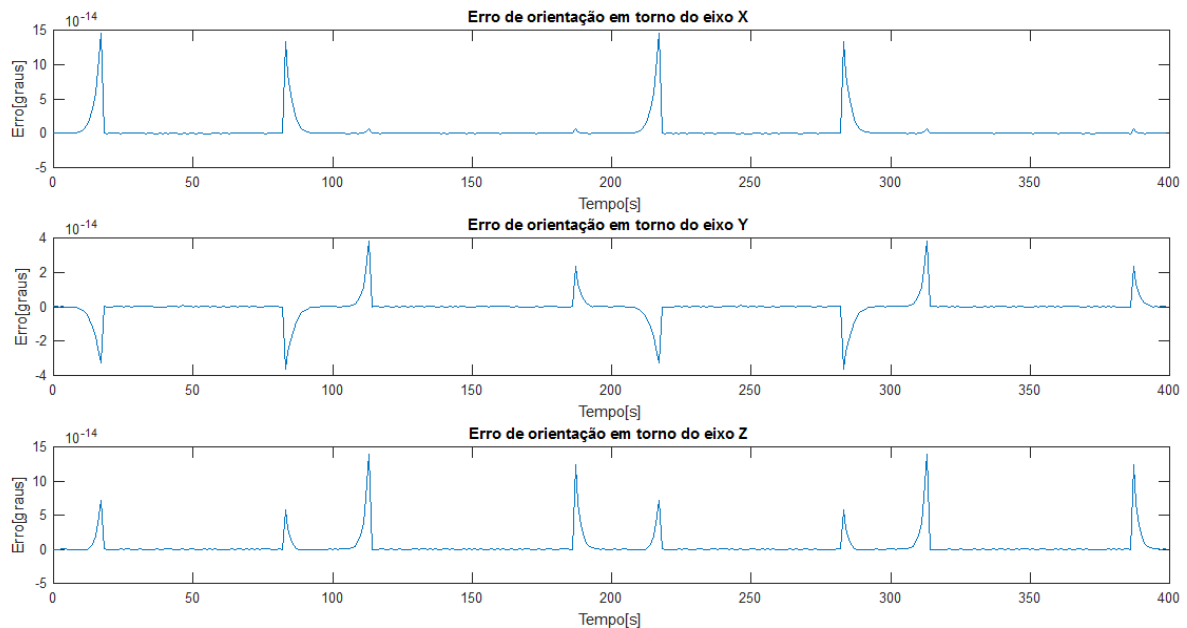


Figura 63 - Erro de orientação no método numérico.



Uma vez que os modelos transcritos para software são validados, o valor para o comprimento de cada atuador linear, obtido como retorno do modelo cinemático inverso, deve ser convertido para ângulo, que ao ser combinado a um sistema biela-manivela, gera movimento similar ao obtido com o atuador linear.

Estruturou-se então um código para gerar o movimento desejado e apresentá-lo graficamente em tempo real, tanto para o mecanismo utilizando atuadores lineares quanto para o mecanismo com atuadores angulares e biela-manivela, por meio das ferramentas gráficas fornecidas pelo MATLAB. O código foi dividido em funções, para facilitar sua visualização e compreensão. Ele também se encontra no Apêndice 4.

Para efetivamente atuar nos motores, este *software* foi escrito em linguagem C++, trabalhando conjuntamente com o *software open source*, fornecido pelo fabricante dos motores. Quando determinado o ângulo de cada atuador, diferentemente do que é feito no MATLAB, em que apresenta-se o resultado graficamente, no código em C++ aplica-se uma conversão deste valor, dado em graus, para unidades de motor e, a partir da resolução deste, monta-se um pacote com a instrução “WRITE”. O endereço inicial de escrita é igual a 30, referente ao byte menos significativo do campo de memória de posição, composto por 2 bytes. Há também a necessidade de declarar os ID’s referentes a cada motor, de forma a direcionar o pacote correto para cada um deles.

Para auxiliar na compreensão do que é feito no código principal, criou-se uma biblioteca chamada “Channel” e um *namespace* “u2d”. Ambos tornam o *software* de comunicação de alto nível e restringem as instruções às de leitura e escrita, transformando todo o código fornecido pelo fabricante em um código de baixo nível, facilitando a visualização do código em geral. Sendo assim, a esta biblioteca são agrupadas todas as outras fornecidas pelo fabricante.

Para tornar este código um executável, usa-se de uma ferramenta chamada “Cmake”. É amplamente conhecido o uso de *makefiles* para facilitar a compilação de códigos em que há certo nível de complexidade, seja este na agregação de bibliotecas a outras bibliotecas, na agregação destas bibliotecas ao código principal, ou até mesmo na geração de executáveis simultâneos que fazem uso de um mesmo conjunto de bibliotecas. Quando esta complexidade se torna ainda mais elevada, de forma que até mesmo a criação de um *makefile* torna-se complexa, é de boa prática a utilização do “Cmake” como uma alternativa. Este *software* é capaz de gerar o *makefile* necessário para compilar códigos a partir de algumas poucas informações presentes em um *cmakelist*. O *cmakelist* para o software desenvolvido, bem como os comentários sobre a função de cada parte, se encontra no Apêndice 5. Na sequência, no Apêndice 6, pode ser observado o código principal com os modelos cinemáticos implementados, o modelo do movimento, a conversão dos dados em unidades de motor e envio dos pacotes via método da classe “Channel”, que é parte da biblioteca “Channel”. Também no Apêndice 6, encontra-se a biblioteca “Channel” e seu arquivo “.h” correspondente. O *namespace* “u2d” também pode ser encontrado no Apêndice

C, juntamente com os códigos fornecidos pelo fabricante: Funcmovu2d.cpp, USB2Dxl.hpp, Dynamixel.c, Dynamixel.h, Dxl_hal.c, Dxl_hal.h.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

O tema em estudo neste trabalho apresenta certa complexidade, tanto no campo matemático quando no campo físico. Desenvolver plataformas que executem atividades complexas exige certo esforço, suporte financeiro e conhecimento por parte daqueles que o fazem. Durante as atividades que levaram aos resultados aqui apresentados, levou-se em consideração diversas simplificações que acabaram por facilitar o desenvolvimento e a análise dos modelos. Porém, como o objetivo do trabalho era o desenvolvimento de um protótipo simplificado e um projeto simulado, é seguro dizer que estes objetivos foram alcançados.

O protótipo simulado atendeu seu objetivo, sendo capaz de exemplificar os seis graus de liberdade da plataforma móvel e ilustrar o funcionamento de uma Plataforma de Stewart.

Quanto ao modelo simulado, todas as características necessárias a uma Plataforma de Stewart para atender aos requisitos estabelecidos foram determinadas. Uma mobilidade de 300mm de translação em todas as direções e 60° em torno de todos os eixos pode ser atendido por atuadores de 600mm de curso. Para uma movimentação a 20mm/s com carga de 20kg, em uma movimentação senoidal na vertical, cada um dos atuadores deve ser capaz de executar forças de até 100N. A unidade hidráulica capaz de fornecer esta energia aos atuadores deve ter capacidade de 12,5KPa com uma vazão de 0,0089496 m³/s.

Em frente a tamanho desafio, os resultados alcançados no desenvolvimento do trabalho são gratificantes. O conteúdo deste trabalho cria a possibilidade de estudos mais aprofundados no assunto por parte dos colegas de universidade e fundamenta bem o conhecimento até aqui explorado.

São inúmeras as possibilidades de aplicação de uma Plataforma de Stewart. Contudo, para cada aplicação diferente, ferramentas diferentes são necessárias para implementar-se a aplicação. Por exemplo: uma Plataforma de Stewart aplicada a um simulador automotivo exige não só que a plataforma seja capaz de seguir uma trajetória desejada, mas também que esta trajetória seja gerada a partir da interação do usuário com o ambiente fictício do simulador. Este tipo de aplicação com *front-end* não era foco deste trabalho, mas é parte importante da utilização efetiva da Plataforma de Stewart.

Outra possibilidade é o uso industrial. Neste caso, as atividades são geralmente predefinidas, mas a velocidade de execução e capacidade de repetição é o desafio. Os modelos matemáticos e físicos costumam não ser os limitadores para estas aplicações, porém a relação do custo/benefício entre eficiência e preço pode ser, uma vez que os equipamentos que resultam em maior velocidade e repetibilidade também

resultam em maior custo de confecção. No ambiente industrial, custo elevado é uma barreira para comercialização.

6.2 PROJETOS FUTUROS

Para continuar o desenvolvimento do trabalhado aqui iniciado, deve-se ainda cotar os componentes que atendem as especificações aqui estabelecidas, especificar os componentes do circuito de controle e, caso seja necessário, projetar aqueles componentes que não forem encontrados no mercado. Uma vez que todos os componentes tenham sido adquiridos, a Plataforma de Stewart pode ser construída. Na sequência, deve-se definir qual a atividade fim da plataforma, uma vez que o tipo de controle, a aplicação front-end e a geração de movimentos dependerão da aplicação.

Para aplicações de alta complexidade, em que a capacidade de carga seja um fator importante, deve-se avaliar a capacidade de carga dos atuadores para a movimentação necessária. Além disso, o controle implementado não é adaptativo, sendo eficaz apenas para o movimento gerado no trabalho. Para atender a vários tipos de movimento, faz-se necessário o desenvolvimento de controladores mais robustos.

REFERÊNCIAS

- [1] ALFARO, M.E., BOLNICK, D.I., WAINWRIGHT, P.C. **Evolutionary dynamics of complex biomechanical systems**: an example using the four-bar mechanism. *Evolution* 58, 495-503, 2004.
- [2] BIANCHI, R. A. C. **Cinemática Direta para manipuladores**. Notas de aula do mestrado em Inteligência Artificial Aplicada a Automação, Centro Universitário da FEI, 2011. Disponível em: <<http://fei.edu.br/~rbianchi/robotica>>. Acesso em: 27/10/2017
- [3] BONEV, I.A.; ZLATANOV, D.; GOSSELIN, C.M. **Singularity Analysis of 3-DOF Planar Parallel Mechanisms via Screw Theory** - *Journal of Mechanical Design*, vol 125 (pg. 573-581) - ASME, 2003
- [4] CABRAL, E. L.L. **Robôs Industriais**. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, Escola Politécnica Universidade de São Paulo, 2004. Disponível em: <<http://www.poli.usp.br/p/eduardo.cabral>>. Acesso em 27/10/2017
- [5] COELHO, T.A.H. **Topologia, Síntese e Análise de uma Estrutura Cinemática Paralela**. Tese de Livre Docência - Escola Politécnica, Universidade de São Paulo. São Paulo, 2005.
- [6] DRGRAEME – **New NXT 2.0 Retail**. Disponível em: <<http://drgraeme.net/DrGraeme-free-NXT-G-tutorials/WhichNXT/Lego-MindStorms-NXT-2.0-Retail/NXT-2.0-Retail-Kit-Lego-MindStorms.htm>>. Acesso em 24/06/2017
- [7] HERSON, T.– **Stewart Platform**. Disponível em: <<http://www.instructables.com/id/Stewart-Platform/>>. Acesso em 24/06/2017
- [8] HO, PAUL T.P.; ET AL. **The Yuan Tseh Lee AMiBA Project**, *Mod. Phys. Lett. A* 23, 1243 (2008)
- [9] PHELAN, R.M., 1962. **Fundamentals of mechanical design**. 2nd ed., McGraw-Hill Book Co, London.
- [10] K. LIU, E L. LEWIS, AND M. FITZGERALD. **Solution Of Nonlinear Kinematics Of A Parallel-Link Constrained Stewart Platform Manipulator**, Vol. 13, No. 2-3, 1994, Pp. 167-183
- [11] MATHWORKS - **Creating a Stewart Platform Model Using SimMechanics**, 2002 Disponível em: <<https://www.mathworks.com/company/newsletters/articles/creating-a-stewart-platform-model-using-simmechanics.html?requestedDomain=www.mathworks.com>>. Acesso em 24/06/2017
- [12] MATHWORKS - **Stewart Platform with Control Reference Trajectory**. Disponível em:<<http://www.mathworks.com/help/phymod/sm/examples/stewart-platform-with-control-reference-trajectory.html>>. Acesso em 24/06/2017

- [13] NORTON, R.L. **Design of Machinery**: An Introduction to the Synthesis and Analysis of Mechanisms and Machines - Second Edition. New Your. MacGraw-Hill, 2001.
- [14] OGATA, K.; **Engenharia de Controle Moderno** - 4ª Edição, 4ª. Ed., Pearson/Prentice Hall, 2005.
- [15] PRABJOT NANUA, KENNETH J. WALDRON AND VASUDEVA MURTHY. **IEEE Transactions On Robotics And Automation**: - Direct Kinematic Solution of a Stewart Platform, Vol. 6. No. 4. August 1990
- [16] ROBOTIS - **e-Manual v1.29.00**: AX-12/ AX-12+/ AX-12. Disponível em: <http://support.robotis.com/en/product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm> . Acesso em 24/06/2017
- [17] ROBOTIS - **e-Manual v1.31.02**: Dynamixel SDK for Linux. Disponível em: <http://support.robotis.com/en/software/dynamixel_sdk/usb2dynamixel/usb2dxl_linux.htm>. Acesso em 24/06/2017
- [18] ROBOTIS - **e-Manual v1.31.02**: API Reference. Disponível em:<http://support.robotis.com/en/software/dynamixel_sdk/api_reference.htm>. Acesso em 24/06/2017
- [19] ROBOTIS - **e-Manual v1.31.02**: Gcc. Disponível em:<http://support.robotis.com/en/software/dynamixel_sdk/usb2dynamixel/linux/gcc.htm>. Acesso em 24/06/2017
- [20] ROBOTIS - **e-Manual v1.29.00**: Kind of Instructions. Disponível em: <http://support.robotis.com/en/product/actuator/dynamixel/communication/dxl_instruction.htm>. Acesso em 24/06/2017
- [21] ROSÁRIO, J. M. **Robótica Industrial I**: Modelagem, Utilização e Programação. São Paulo: Barauna, 2010
- [22] SELEGATTO, L. T.; **Análise dinâmica de um mecanismo biela-manivela com folgas nas juntas pino-pistão e pistão-cilindro**, Campinas, SP, 2016.
- [23] SOUZA, C. R.; **Avaliação do Desempenho de Máquinas de Usinagem com Estrutura Cinemática Paralela Plana**. Tese de Mestrado - Escola Politécnica, Universidade de São Paulo. São Paulo, 2007.
- [24] TAVARES, P. A; PINTO, N. B. R.; **Análise De Movimento Para Um Sistema Biela Manivela**, 2014. Disponível em: <http://ppstma.unievangelica.edu.br/sncma/anais/anais/2014/2014_painel_008.pdf>. Acesso em 15/10/2017
- [25] USP: UNIVERSIDADE DE SÃO PAULO. LDSV - Laboratório de Dinâmica e Simulação Veicular - **Plataforma Móvel 3D**. Disponível em: <http://www.usp.br/ldsv/?page_id=383>. Acesso em 15/10/2017

[26] WENDLANDT, J. – **Stewart Platform Mechanical System**. Disponível em: <[https:// www.mathworks.com/matlabcentral/fileexchange/2334-stewart-platform-mechanical-system](https://www.mathworks.com/matlabcentral/fileexchange/2334-stewart-platform-mechanical-system)>. Acesso em 24/07/2017

[27] WUYTENS, D. – **Universal Joint LEGO**. Disponível em: <<https://grabcad.com/library/universal-joint-lego>>. Acesso em 24/07/2017

[28] YAUHEN. – **Lego Technic Axles**. Disponível em: <<https://grabcad.com/library/lego-technic-axles-1>>. Acesso em 24/07/2017

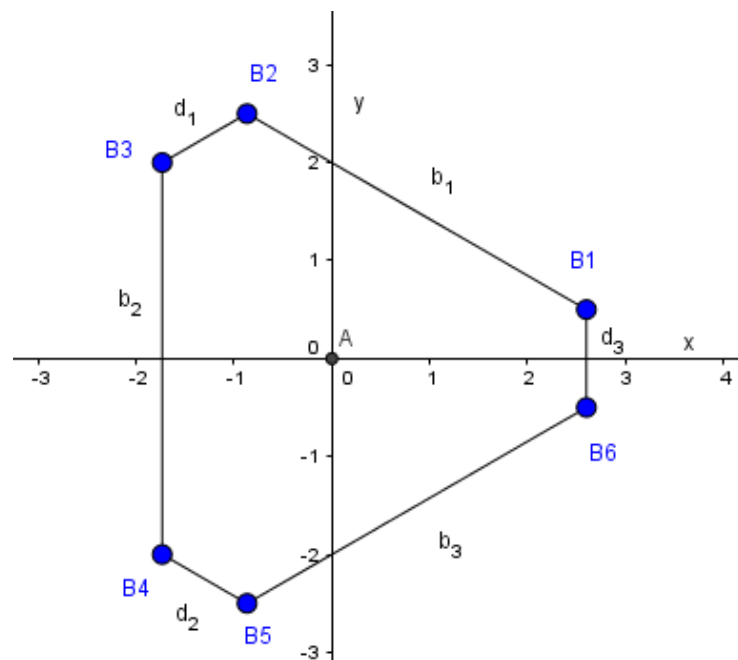
APÊNDICES

APÊNDICE 1 - Solução de cinemática direta não linear de uma plataforma de stewart

O modelo cinemático direto para esta abordagem é obtido a partir de restrições geométricas às plataformas superior e inferior. A plataforma inferior deve ser um hexágono semi-regular, enquanto a plataforma superior deve ser um triângulo equilátero.

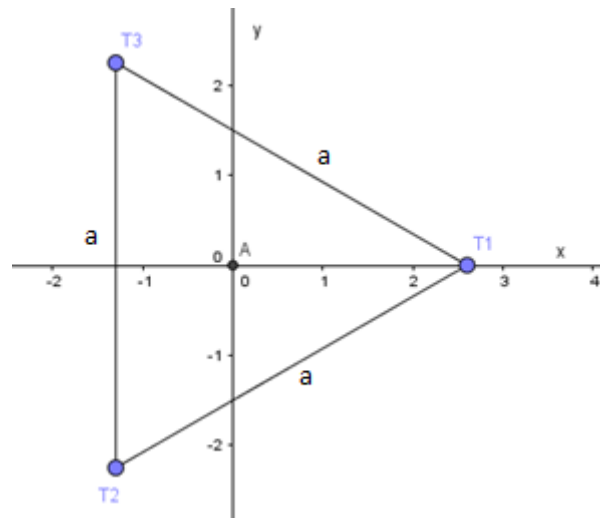
Com a geometria definida, têm-se a liberdade de escolher as dimensões b_i, d_i (para $i = 1..3$) e a , que dizem respeito a maior aresta do hexágono semi-regular, a menor aresta do hexágono semi-regular e a aresta do triângulo equilátero, respectivamente. Uma visão esquemática é apresentada nas Figuras 64 e 65.

Figura 64 - Plataforma inferior.



Fonte: elaborado pelo autor.

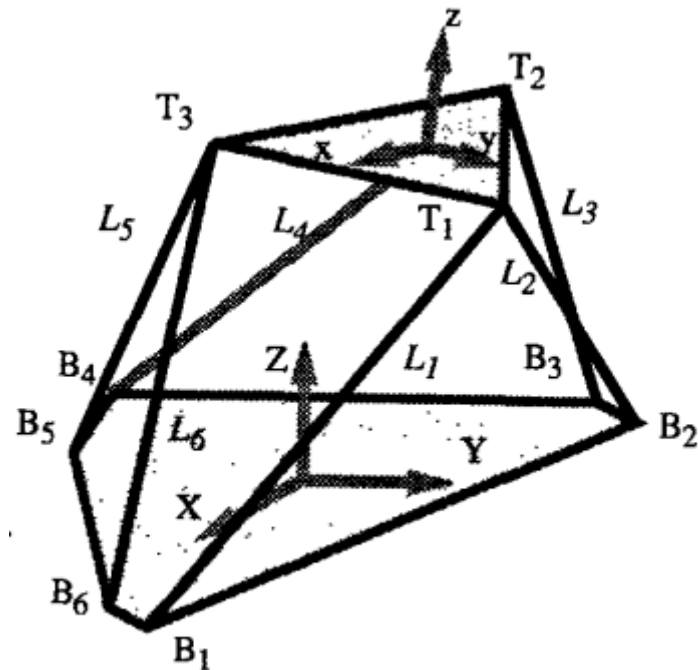
Figura 65 - Plataforma superior.



Fonte: elaborado pelo autor.

Considerando o comprimento dos atuadores em um dado instante como L_1, L_2, L_3, L_4, L_5 e L_6 , com um sistema de coordenadas inercial (X, Y, Z) fixado no centro de massa da plataforma inferior e um sistema de coordenadas móvel fixado no centro de massa da plataforma superior (x, y, z) , denota-se, com relação ao sistema de coordenadas inercial, os pontos T_1, T_2, T_3 como sendo os vértices da plataforma superior, bem como os pontos $B_1, B_2, B_3, B_4, B_5, B_6$ como sendo os vértices da plataforma inferior, conforme mostra a Figura 66 na sequência.

Figura 66 - Sistema de coordenadas.

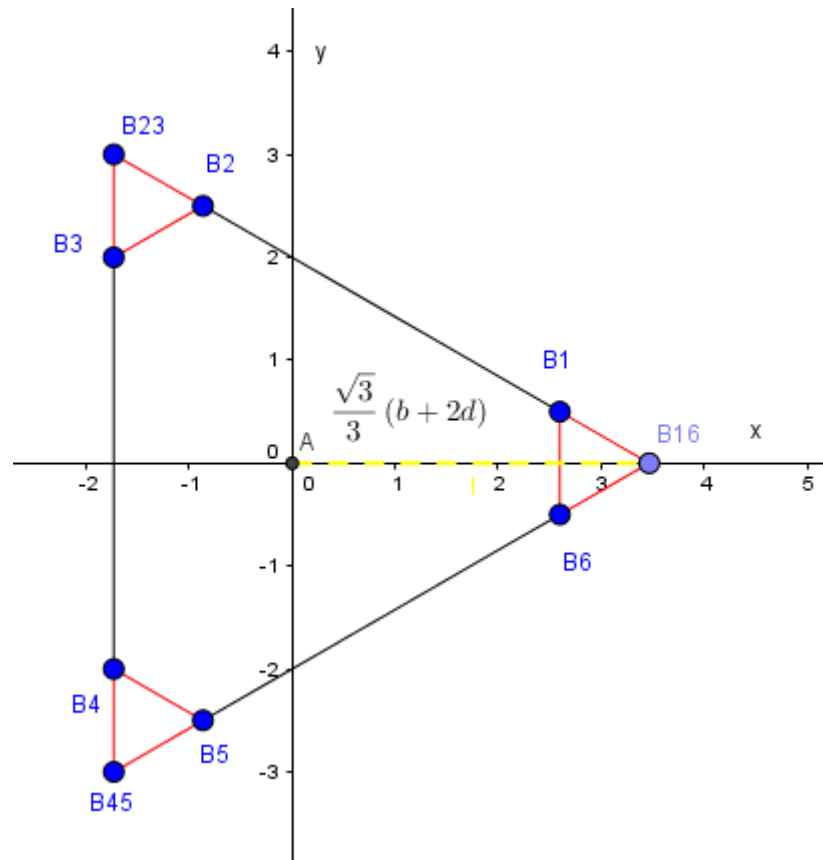


Fonte: (LIU; LEWIS; FITZGERALD, 1994)

Uma vez estabelecido o sistema de coordenadas inercial, deve-se descrever cada um dos pontos com relação a este. Para alcançar tal objetivo, utiliza-se de análises trigonométricas que, devido ao formato da plataforma, facilitam a determinação das coordenadas de cada ponto.

Conforme observado na Figura 67, combinando o hexágono semi-regular a triângulos equiláteros (em vermelho), obtém-se um grande triângulo equilátero, cuja posição do centro de massa, considerando o material de densidade uniforme, pode ser dado pela magnitude da linha amarela pontilhada, utilizando a equação apresentada também na Figura 67.

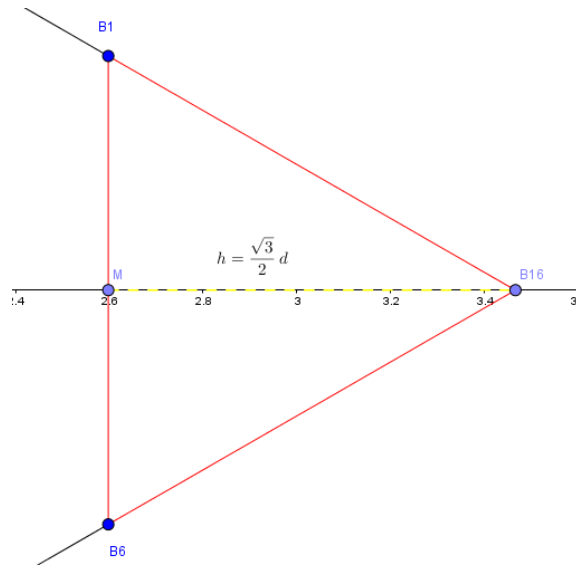
Figura 67 - Plataforma Inferior como triângulo equilátero.



Fonte: elaborado pelo autor.

Por outro lado, a altura de um triângulo retângulo pode ser dada conforme mostrado na Figura 68. A altura apresentada é referente ao triângulo vermelho utilizado para completar a forma geométrica do hexágono semi-regular.

Figura 68 - Altura do triângulo equilátero.



Fonte: elaborado pelo autor.

Com essas informações, determina-se os pontos B1 a B6 da plataforma inferior. Começando pelo ponto B1, a sua coordenada X_{B1} pode ser determinada pela distância do ponto B16 ao centro de massa, ou seja, a origem do sistema de coordenadas conforme definido na Figura 67, dada pela equação do centro de massa de triângulos equiláteros, subtraída da altura do triângulo equilátero B6B16B1.

$$X_{B1} = \frac{\sqrt{3}}{3}(b + 2d) - \frac{\sqrt{3}}{2}(d) \quad (6)$$

$$X_{B1} = \frac{\sqrt{3}}{6}(2b + 4d - 3d) \quad (7)$$

$$X_{B1} = \frac{\sqrt{3}}{6}(2b + d) \quad (8)$$

Observando-se a Figura 67, deduz-se que a coordenada Y_{B1} é dada por:

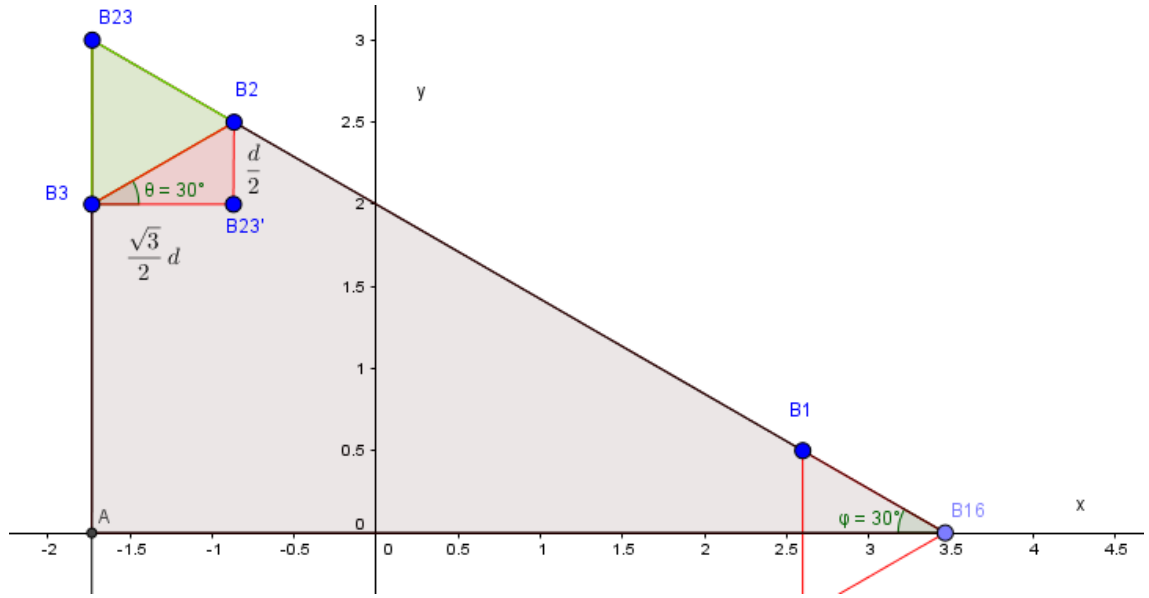
$$Y_{B1} = \frac{1}{2}d \quad (9)$$

E, como o ponto está no mesmo plano que o sistema de coordenadas, a coordenada Z_{B1} é dada por:

$$Z_{B1} = 0 \quad (10)$$

Para determinar-se as coordenadas do ponto B2, a plataforma inferior deve ser analisada conforme mostrado na Figura 69.

Figura 69 - Determinação do ponto B2.



Fonte: elaborado pelo autor.

A coordenada X_{B2} é dada pela altura do triângulo B16B23B45, conforme visto na Figura 67, subtraído da posição do centro de massa, somado ao cateto adjacente ao ângulo θ do triângulo em vermelho na Figura 69.

$$X_{B2} = \frac{\sqrt{3}}{3}(b + 2d) - \frac{\sqrt{3}}{2}(b + 2d) + \frac{\sqrt{3}}{2}(d) \quad (11)$$

$$X_{B2} = \frac{\sqrt{3}}{3}(b + 2d) - \frac{\sqrt{3}}{2}(b + d) \quad (12)$$

$$X_{B2} = \frac{\sqrt{3}}{6}(2b + 4d - 3b - 3d) \quad (13)$$

$$X_{B2} = -\frac{\sqrt{3}}{6}(b - d) \quad (14)$$

A coordenada Y_{B2} é dada por metade do valor da maior aresta do hexágono semi-regular somada ao cateto oposto ao ângulo θ do triângulo em vermelho na Figura 69.

$$Y_{B2} = \frac{b}{2} + \frac{d}{2} \quad (15)$$

$$Y_{B2} = \frac{1}{2}(b + d) \quad (16)$$

Na forma como visto anteriormente, uma vez que o ponto está no plano XY, a coordenada Z_{B2} é dada por:

$$Z_{B2} = 0 \quad (17)$$

Em análise semelhante à feita para obtenção das coordenadas do ponto B2, pode-se obter as coordenadas para o ponto B3. O que os difere é a ausência da

influência do triângulo vermelho, da Figura 69, na magnitude das coordenadas do ponto. Sendo assim, as coordenadas do ponto B3 são:

$$X_{B3} = \frac{\sqrt{3}}{3}(b + 2d) - \frac{\sqrt{3}}{2}(b + 2d) \quad (18)$$

$$X_{B3} = \frac{\sqrt{3}}{6}(2b + 4d - 3b - 6d) \quad (19)$$

$$X_{B3} = -\frac{\sqrt{3}}{6}(b + 2d) \quad (20)$$

$$Y_{B3} = \frac{1}{2}b \quad (21)$$

$$Z_{B3} = 0 \quad (22)$$

O ponto B4, conforme a Figura 67, se assemelha ao ponto B3, diferindo apenas no sinal da coordenada Y. As coordenadas do ponto B4 são:

$$X_{B4} = -\frac{\sqrt{3}}{6}(b + 2d) \quad (23)$$

$$Y_{B4} = -\frac{1}{2}b \quad (24)$$

$$Z_{B4} = 0 \quad (25)$$

O ponto B5, conforme a Figura 67, se assemelha ao ponto B2, diferindo apenas no sinal da coordenada Y. As coordenadas do ponto B5 são:

$$X_{B5} = -\frac{\sqrt{3}}{6}(b - d) \quad (26)$$

$$Y_{B5} = -\frac{1}{2}(b + d) \quad (27)$$

$$Z_{B5} = 0 \quad (28)$$

O ponto B6, conforme a Figura 67, se assemelha ao ponto B1, diferindo apenas no sinal da coordenada Y. As coordenadas do ponto B6 são:

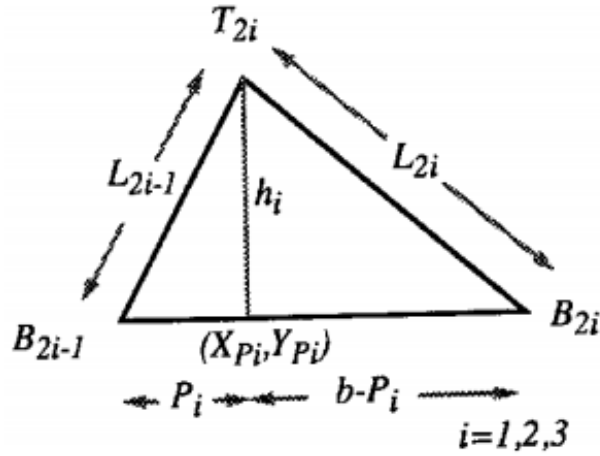
$$X_{B6} = \frac{\sqrt{3}}{6}(2b + d) \quad (29)$$

$$Y_{B6} = -\frac{1}{2}d \quad (30)$$

$$Z_{B6} = 0 \quad (31)$$

Determinado os pontos da plataforma superior, o próximo passo é determinar os pontos T1, T2 e T3, da plataforma superior. Porém, estes pontos dependem da configuração dos atuadores L1, L2, L3, L4, L5 e L6. Dessa forma, é preciso encontrar a relação entre estes para determinar os pontos desejados em função do sistema de coordenadas inercial.

Figura 70 - Triângulo formado pelos atuadores.



Fonte: (LIU; LEWIS; FITZGERALD, 1994)

Sendo L_{2i} e L_{2i-1} um par qualquer de atuadores conectados à plataforma inferior nos pontos B_{2i} e B_{2i-1} respectivamente, e ao ponto T_{2i} na plataforma superior, conforme representado pelo triângulo na Figura 70, a projeção sobre o segmento $\overline{B_{2i}B_{2i-1}}$ dos atuadores L_{2i} e L_{2i-1} resultará em dois segmentos de reta com origem dos pontos B_{2i} e B_{2i-1} e fim no ponto P_i . Escrevendo a altura h_i em função de L_{2i} , L_{2i-1} e b (o mesmo que o segmento $\overline{B_{2i}B_{2i-1}}$, ou seja, aresta do hexágono semi-regular), tem-se que a magnitude $\|B_{2i-1}P_i\|$ é dado por:

$$h_i^2 = L_{2i-1}^2 - \|B_{2i-1}P_i\|^2 = L_{2i}^2 - (b - \|B_{2i-1}P_i\|)^2 \quad (32)$$

$$L_{2i-1}^2 - \|B_{2i-1}P_i\|^2 = L_{2i}^2 - \|B_{2i-1}P_i\|^2 + 2\|B_{2i-1}P_i\|b - b^2 \quad (33)$$

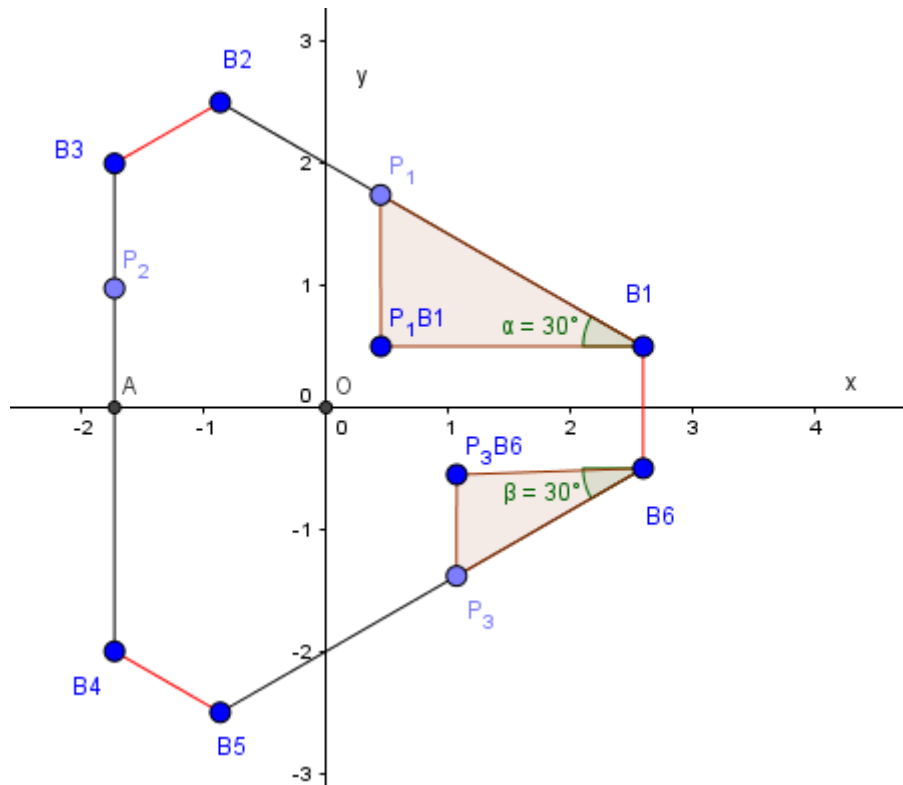
$$L_{2i-1}^2 = L_{2i}^2 + 2\|B_{2i-1}P_i\|b - b^2 \quad (34)$$

$$2\|B_{2i-1}P_i\|b = -L_{2i}^2 + L_{2i-1}^2 + b^2 \quad (35)$$

$$\|B_{2i-1}P_i\| = \frac{1}{2b} (+L_{2i-1}^2 - L_{2i}^2 + b^2) \quad (36)$$

O módulo será necessário para determinar as coordenadas do ponto P_i com relação ao sistema de coordenadas inercial, e conforme visto na Figura 71, também representa o módulo dos segmentos $\overline{B_6P_3}$, $\overline{B_1P_1}$ e $\overline{B_3P_2}$, sendo que os 2 primeiros são as hipotenusas dos triângulos mostrados na Figura 71.

Figura 71 - Coordenadas dos pontos P_i .



Fonte: elaborada pelo autor.

As coordenadas X e Y dos pontos B1 e B6 são conhecidas. Nota-se que a coordenada X dos pontos P_1 e P_3 é equivalente à coordenada X dos pontos B1 e B6 subtraída pelos valores dos catetos $\overline{P_1B1}$ e $\overline{P_3B6}$, respectivamente. Análise similar pode ser feita para afirmar que a coordenada Y dos pontos P_1 e P_3 é dada pela coordenada Y dos pontos B1 e B6, somadas aos valores dos catetos $\overline{P_1B1}$ e $\overline{P_3B6}$, respectivamente. Sendo assim, pode-se escrever as coordenadas de tais pontos como:

$$X_{P1} = X_{B1} - \frac{\sqrt{3}}{2} ||P_1B1|| \quad (37)$$

$$Y_{P1} = Y_{B1} + \frac{1}{2} ||P_1B1|| \quad (38)$$

$$Z_{P1} = 0 \quad (39)$$

$$X_{P3} = X_{B6} - \frac{\sqrt{3}}{2} ||P_3B6|| \quad (40)$$

$$Y_{P3} = Y_{B6} - \frac{1}{2} ||P_3B6|| \quad (41)$$

$$Z_{P3} = 0 \quad (42)$$

Quanto a P_2 , a coordenada X é igual à dos pontos B3 e B4, e a coordenada Y pode ser dada pela diferença entre o valor da coordenada Y do ponto B3 e o valor do módulo de $\overline{P_2B3}$. Sendo assim, pode-se escrever as coordenadas da seguinte forma:

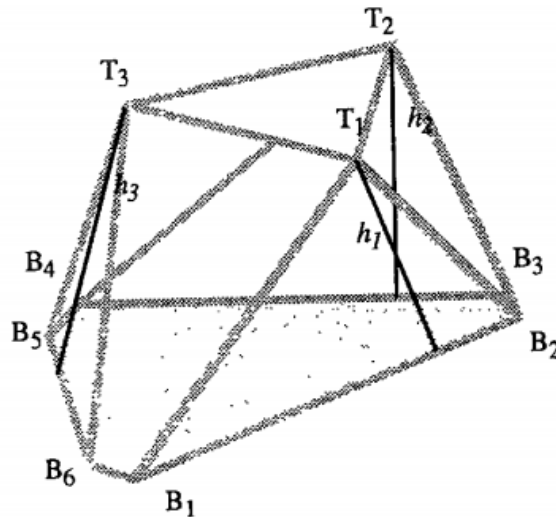
$$X_{P2} = X_{B3} = X_{B4} \quad (43)$$

$$Y_{P2} = Y_{B3} - ||P_2B3|| \quad (44)$$

$$Z_{P2} = 0 \quad (45)$$

Estes triângulos, como os da Figura 70, não precisam necessariamente ser perpendiculares à plataforma inferior, como pode ser visto na Figura 72 que segue.

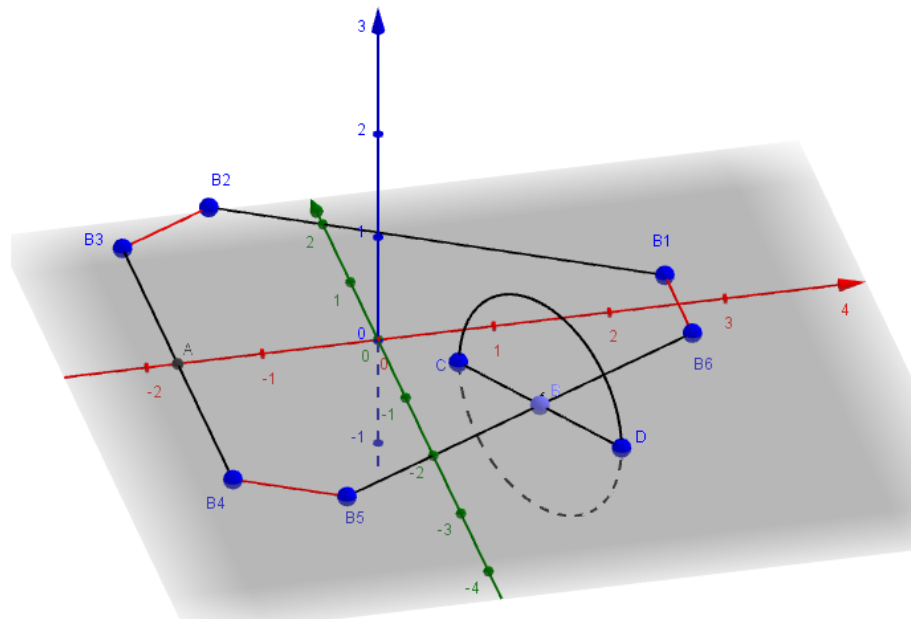
Figura 72 - Plataforma com triângulos formados por atuadores.



Fonte: (LIU; LEWIS; FITZGERALD, 1994)

Imaginando que não houvesse nenhuma restrição ao movimento de um ponto qualquer T_{2i} da plataforma superior, seria possível que, para dadas dimensões dos atuadores L_{2i} e L_{2i-1} , o ponto girasse em torno de uma circunferência de raio h_i , centrada em algum ponto do segmento $\overline{B_{2i}B_{2i-1}}$, como visto na Figura 73.

Figura 73- Possíveis pontos para T_{2i} .

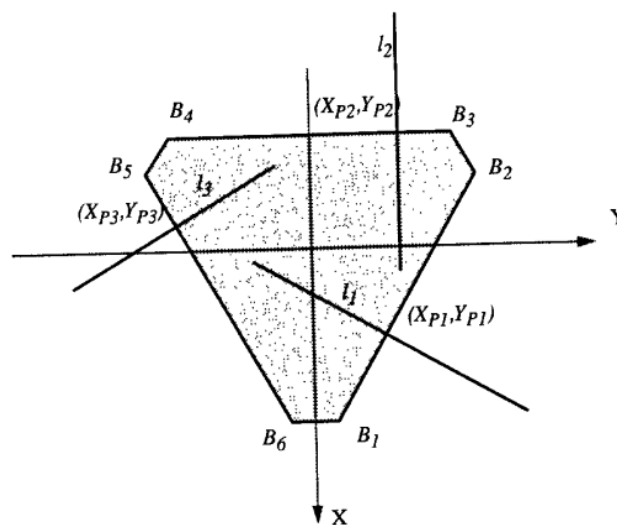


Fonte: elaborada pelo autor.

Neste caso, a projeção do ponto T_{2i} no plano de coordenadas inercial X e Y, será sempre um ponto sobre o segmento $\overline{CD}$.

Sendo assim, para uma dada configuração dos atuadores, as retas que descrevem as possíveis projeções no plano XY para quaisquer dos pontos T_{2i} da plataforma superior podem ser vistas na Figura 74.

Figura 74 - Retas de possíveis projeções de T_{2i} .

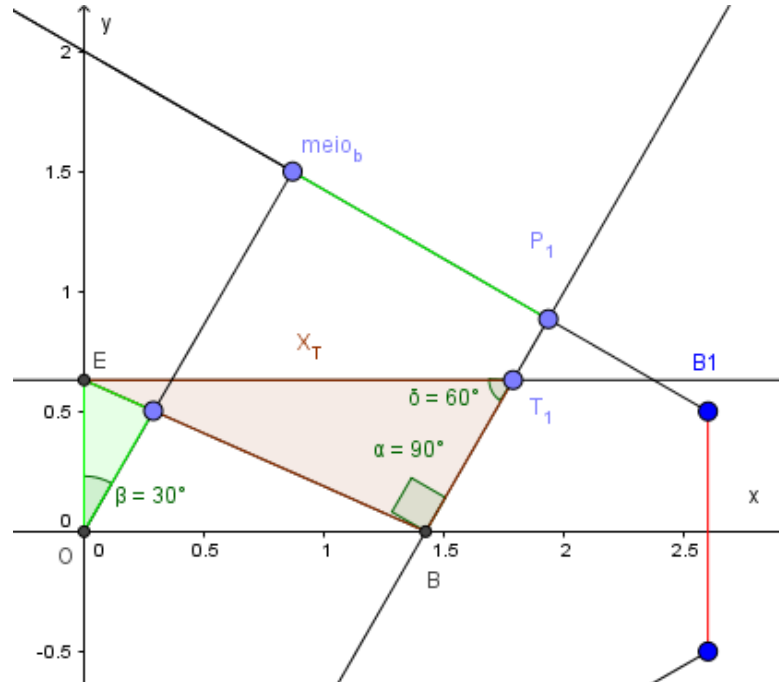


Fonte: (LIU; LEWIS; FITZGERALD, 1994)

Tais retas são perpendiculares aos segmentos formados pelas arestas da base inferior, como pode ser visto na Figura 72. Analisando o ponto T_1 em uma posição

qualquer sobre esta reta, como visto na Figura 75, pode-se escrever a coordenada Y_{T_1} em função da coordenada X_{T_1} .

Figura 75 - Y_{T_1} em função de X_{T_1} .



Fonte: elaborada pelo autor.

Observando os triângulos formados na Figura 75, nota-se que, conforme foram definidos, a hipotenusa do triângulo BET_1 é paralela ao eixo X, sendo o segmento módulo do $\overline{ET_1}$ o mesmo que a coordenada X do ponto T_1 . Sendo assim, o cateto oposto ao ângulo de 60 graus no triângulo BET_1 é dado por:

$$C_{op_{BET_1}} = \frac{\sqrt{3}}{2} X_{T_1} \quad (46)$$

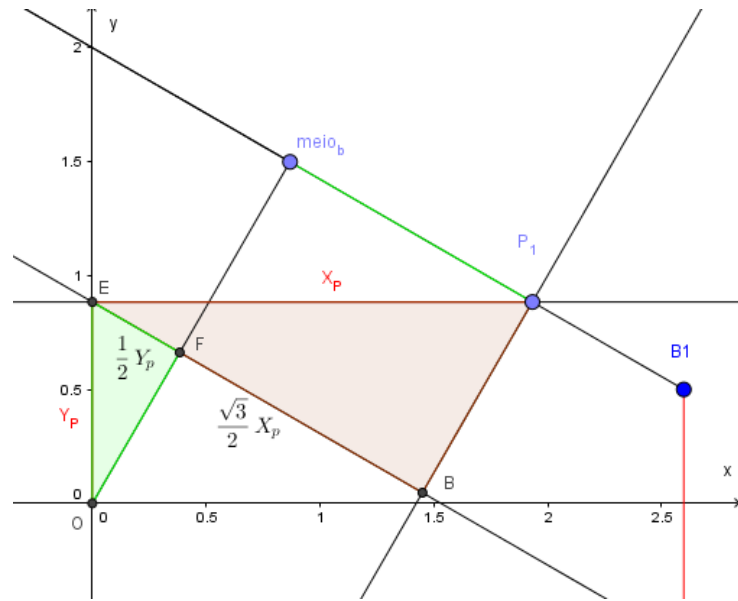
Subtraindo de $b/2$ o módulo do segmento $\overline{B_1P_1}$, obtém-se o módulo do segmento $\overline{meio_bP_1}$, em verde na Figura 75. É válido lembrar que o cateto adjacente deste mesmo triângulo BET_1 está sobre a reta dos possíveis pontos da projeção de T_1 , que conforme já visto anteriormente, é perpendicular a $\overline{B_1B_2}$, segmento do qual $\overline{meio_bP_1}$ faz parte. Logo, $\overline{meio_bP_1}$ é perpendicular a este cateto adjacente, que por sua vez, é perpendicular $C_{op_{BET_1}}$, tornando então $C_{op_{BET_1}}$ paralelo a $\overline{meio_bP_1}$. Subtraindo o módulo de $\overline{meio_bP_1}$ de $C_{op_{BET_1}}$, obtém-se o valor do cateto oposto ao ângulo de 30 graus (β) do triângulo verde, mostrado na Figura 75. Este triângulo, formado por um segmento de reta que passa por $meio_b$, paralelo à reta dos possíveis pontos da projeção de T_1 , e parte do segmento referente ao $C_{op_{BET_1}}$, é retângulo. O fato pode ser constatado por uma análise dos ângulos alternos internos referentes aos dois triângulos e o ângulos de 90 graus do triângulo BET_1 . A hipotenusa deste triângulo é equivalente à coordenada Y_{T_1} . Sendo assim, pode-se escrever Y_{T_1} na forma:

$$Y_{T1} \frac{1}{2} = C_{op_{verde}} = \frac{\sqrt{3}}{2} X_{T1} - ||meio_b P_1|| \quad (46)$$

$$Y_{T1} \frac{1}{2} = C_{op_{verde}} = \frac{\sqrt{3}}{2} X_{T1} - \frac{b}{2} + ||P_1 B1|| \quad (47)$$

Porém, o módulo do segmento $\overline{meio_b B1}$ pode ser escrito de outra forma, utilizando-se uma relação trigonométrica muito semelhante à esta última. Conforme pode ser visto na Figura 76, pode-se escrever este módulo em função das coordenadas X e Y do ponto P_1 .

Figura 76 - Reescrevendo o segmento.



Fonte: elaborada pelo autor.

Assim, pode-se escrever o módulo do segmento como:

$$||meio_b P_1|| = \frac{\sqrt{3}}{2} X_{P1} - \frac{1}{2} Y_{P1} = -\frac{b}{2} + ||P_1 B1|| \quad (48)$$

Logo, Y_{T1} pode ser escrito como:

$$Y_{T1} \frac{1}{2} = C_{op_{verde}} = \frac{\sqrt{3}}{2} X_{T1} - ||meio_b P_1|| \quad (49)$$

$$Y_{T1} \frac{1}{2} = \frac{\sqrt{3}}{2} X_{T1} - \frac{\sqrt{3}}{2} X_{P1} + \frac{1}{2} Y_{P1} \quad (50)$$

$$Y_{T1} = \sqrt{3} X_{T1} - (\sqrt{3} X_{P1} - Y_{P1}) \quad (51)$$

Analogamente, obtém-se Y_{T3} em função de X_{T3} :

$$Y_{T3} \frac{1}{2} = -\frac{\sqrt{3}}{2} X_{T3} - \frac{b}{2} + ||P_3 B_6|| \quad (51)$$

Então, pode-se escrever o ponto Y_{T3} da seguinte forma:

$$Y_{T3} \frac{1}{2} = -\frac{\sqrt{3}}{2} X_{T3} + \frac{\sqrt{3}}{2} X_{P3} + \frac{1}{2} Y_{P3} \quad (52)$$

$$Y_{T3} = -\sqrt{3} X_{T3} + \sqrt{3} X_{P3} + Y_{P3} \quad (53)$$

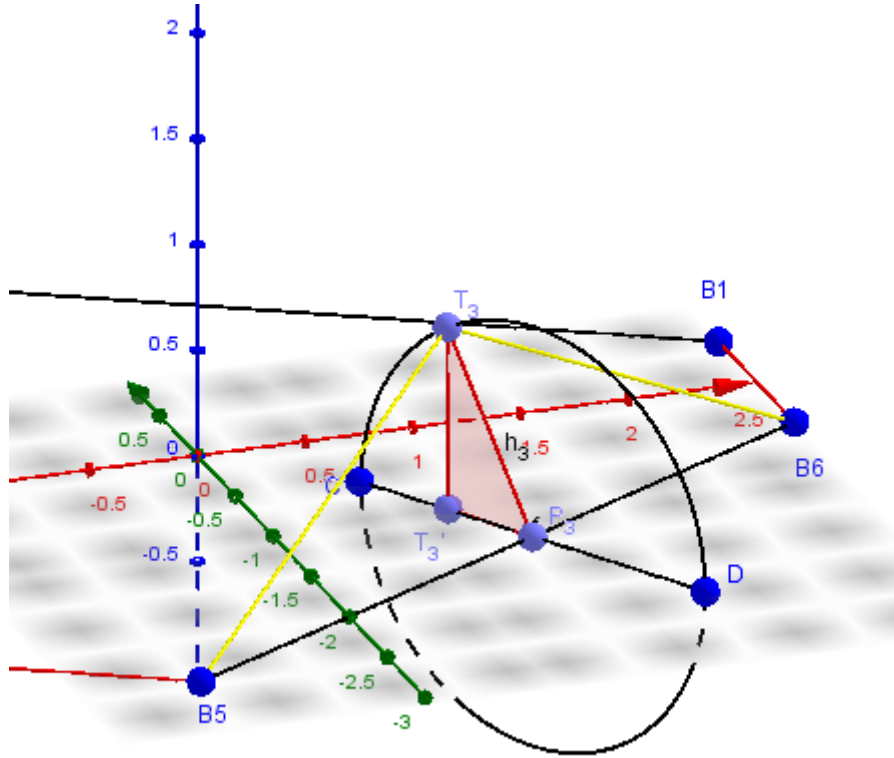
Sendo as diferenças de sinais fruto dos valores negativos das coordenadas Y_{P3} e Y_{T3} .

Por fim, conforme visto na Figura 74, o valor de Y_{T2} vai ser sempre o mesmo de Y_{P2} .

$$Y_{T2} = Y_{P2} \quad (54)$$

As projeções no plano nada mais são que as coordenadas X e Y dos pontos da plataforma superior. Uma vez que todas as coordenadas Y dos pontos da plataforma superior estão descritas, ou em função de suas respectivas coordenadas X, ou em função de variáveis já conhecidas, o próximo passo é fazer o mesmo para as coordenadas Z. Com o auxílio da Figura 77, pode-se observar a formação de um triângulo ligando, nesse exemplo, o ponto T_3 , que agora representa o ponto real na plataforma superior e não mais uma projeção, T'_3 , a projeção de T_3 no plano XY cujos valores das coordenadas X e Y foram calculadas anteriormente, e o ponto P_3 , já determinado.

Figura 77 - Triângulo formado com a projeção do ponto T_3 .



Fonte: elaborada pelo autor.

Como é possível notar, o valor h_3 é conhecido, bem como a distância entre os pontos P_3 e T_3' pode ser determinada, e será função de X_{T_3} . Sendo assim, por Pitágoras, pode-se determinar a magnitude do segmento $\overline{T_3 T_3'}$, que nada mais é do que a coordenada Z do ponto T_3 .

$$Z_{T_3} = \sqrt{h_3^2 - ||T_3' P_3||^2} \quad (55)$$

Sendo que,

$$h_i = \sqrt{L_{2i-1}^2 - ||B_{2i-1} P_i||^2} \quad (56)$$

Como visto na Figura 77, e

$$||T_i' P_i||^2 = (X_{P_i} - X_{T_i'})^2 + (Y_{P_i} - Y_{T_i'})^2 \quad (57)$$

Para evitar confusão, vale lembrar que as coordenadas X e Y de T_i e T_i' são iguais. Aplicando as substituições para os valores de Y_{T_3} e Y_{P_3} , encontra-se que:

$$Z_{T_3} = \sqrt{h_3^2 - \{(X_{P_3} - X_{T_3})^2 + [Y_{P_3} - (-\sqrt{3}X_{T_3} + \sqrt{3}X_{P_3} + Y_{P_3})]^2\}} \quad (57)$$

$$Z_{T_3} = \sqrt{h_3^2 - [(X_{P_3} - X_{T_3})^2 + (+\sqrt{3}X_{T_3} - \sqrt{3}X_{P_3})^2]} \quad (58)$$

$$Z_{T3} = \sqrt{h_3^2 - \{(X_{P3} - X_{T3})^2 + [\sqrt{3}(X_{T3} - X_{P3})]^2\}} \quad (59)$$

$$Z_{T3} = \sqrt{h_3^2 - [(X_{P3} - X_{T3})^2 + 3(-1)^2(X_{P3} - X_{T3})^2]} \quad (60)$$

$$Z_{T3} = \sqrt{h_3^2 - 4(X_{P3} - X_{T3})^2} \quad (61)$$

Com triângulos similares para os demais pontos T_1 e T_2 , e assim como para T_3 , realizando as substituições dos valores já encontrados, e simplificando os resultados, obtém-se:

$$Z_{T2} = \sqrt{h_2^2 - (X_{T2} - X_{P2})^2} \quad (62)$$

$$Z_{T1} = \sqrt{h_1^2 - 4(X_{T1} - X_{P1})^2} \quad (63)$$

Todas as coordenadas dos pontos da plataforma superior estão agora descritas em função de X_{Ti} . Na sequência, aplica-se a restrição dimensional da plataforma superior.

$$||T_1T_2||^2 = ||T_2T_3||^2 = ||T_3T_1||^2 = a^2 \quad (64)$$

O que significa dizer que:

$$(X_{T1} - X_{T2})^2 + (Y_{T1} - Y_{T2})^2 + (Z_{T1} - Z_{T2})^2 = a^2 \quad (65)$$

Ou,

$$(X_{T1} - X_{T2})^2 + (Y_{T1} - Y_{T2})^2 + (Z_{T1} - Z_{T2})^2 - a^2 = 0 \quad (66)$$

Ao substituir os valores das coordenadas Y e Z pelos valores encontrados anteriormente, em função de X_{Ti} , têm-se:

$$\begin{aligned} & (X_{T1} - X_{T2})^2 + [\sqrt{3}X_{T1} - (\sqrt{3}X_{P1} - Y_{P1}) - Y_{P2}]^2 \\ & + \left(\sqrt{h_1^2 - 4(X_{T1} - X_{P1})^2} - \sqrt{h_2^2 - (X_{T2} - X_{P2})^2} \right)^2 - a^2 = 0 \end{aligned} \quad (67)$$

Repetindo o mesmo procedimento para as demais restrições, têm-se:

$$\begin{aligned} & (X_{T2} - X_{T3})^2 + [Y_{P2} - (-\sqrt{3}X_{T3} + \sqrt{3}X_{P3} + Y_{P3})]^2 \\ & + \left(\sqrt{h_2^2 - (X_{T2} - X_{P2})^2} - \sqrt{h_3^2 - 4(X_{T3} - X_{P3})^2} \right)^2 - a^2 = 0 \end{aligned} \quad (68)$$

$$\begin{aligned} & (X_{T3} - X_{T1})^2 + \{-\sqrt{3}X_{T3} + \sqrt{3}X_{P3} + Y_{P3} - [\sqrt{3}X_{T1} - (\sqrt{3}X_{P1} - Y_{P1})]\}^2 \\ & + \left(\sqrt{h_3^2 - 4(X_{T3} - X_{P3})^2} - \sqrt{h_1^2 - 4(X_{T1} - X_{P1})^2} \right)^2 - a^2 = 0 \end{aligned} \quad (69)$$

Resolvendo o conjunto de equações, determina-se os valores das coordenadas X dos pontos da plataforma superior, e como todo o restante dos pontos é função destas coordenadas, consegue-se então descrever todos os pontos da Plataforma de Stewart.

Para resolver o conjunto de equações, optou-se pelo uso de uma ferramenta numérica, devido à complexidade e alto custo computacional decorrentes da solução analítica.

O método numérico é chamado Newton-Raphson e, para implementá-lo, deve-se inicialmente construir uma matriz de Jacobianos, que nada mais é que uma matriz 3×3 contendo, em cada linha, a derivada parcial de cada uma das três funções em relação à cada uma das três variáveis. Considerando a Equação (67) igual a $F1$ e não mais igual a zero, a Equação (68) igual a $F2$ e a Equação (69) igual a $F3$. Considerando também suas derivadas parciais, obtém-se uma matriz de Jacobiano dada por:

$$\begin{bmatrix} \frac{\partial F1}{\partial X_{T1}} & \frac{\partial F1}{\partial X_{T2}} & \frac{\partial F1}{\partial X_{T3}} \\ \frac{\partial F2}{\partial X_{T1}} & \frac{\partial F2}{\partial X_{T2}} & \frac{\partial F2}{\partial X_{T3}} \\ \frac{\partial F3}{\partial X_{T1}} & \frac{\partial F3}{\partial X_{T2}} & \frac{\partial F3}{\partial X_{T3}} \end{bmatrix} = J \quad (70)$$

Para facilitar a compreensão, a matriz resultante deste processo não será apresentada aqui, mas estará contida no *Apêndice 4*, conforme implementado em código no MATLAB.

Construída a matriz, é necessário escolher um valor inicial para as coordenadas X , de forma a possibilitar o início das interações. Como palpite inicial, foi determinado que os valores de X_{T1}, X_{T2}, X_{T3} seriam iguais aos valores X_{P1}, X_{P2} e X_{P3} , respectivamente, pois já são valores conhecidos.

Aplicando este valor inicial às equações, determina-se os valores iniciais de $F1$, $F2$ e $F3$. Constrói-se então um vetor coluna F a partir destes valores e multiplica-se este pela inversa da matriz de Jacobiano. Como resultado, será obtido um vetor linha chamado ΔX .

$$\Delta X = J^{-1} * F \quad (71)$$

Este ΔX representa o erro aproximado entre os valores escolhidos para as coordenadas X e os que supostamente estão próximos da solução exata para o sistema de equações. Cada elemento do vetor diz respeito à uma das três coordenadas X (X_{T1}, X_{T2}, X_{T3}). Sendo assim, para a próxima iteração deve-se atualizar os valores das coordenadas.

$$X_{T1} = X_{T1} + \Delta X[1] \quad (72)$$

$$X_{T2} = X_{T2} + \Delta X[2] \quad (73)$$

$$X_{T3} = X_{T3} + \Delta X[3] \quad (74)$$

Além do ΔX , existe outro valor que pode ser utilizado para avaliar a qualidade do ajuste do processo iterativo. Este valor é dado pelo erro entre o valor encontrado ($F1, F2, F3$), referentes aos valores de X aplicados na interação, e o valor desejado (0,0,0).

Desta forma, existem 2 possíveis critérios de parada: o primeiro é se o valor de ΔX atinge um patamar menor que o estipulado como erro aceitável, o que significa que os valores escolhidos para X_{T1}, X_{T2}, X_{T3} estão com um bom ajuste; o segundo é se os valores de $F1, F2, F3$ estão muito próximos de zero, com erro menor que o estipulado como critério aceitável, o que significa que as equações convergiram para o valor desejado com os últimos valores de X_{T1}, X_{T2}, X_{T3} testados.

O número de iterações necessárias para convergir-se a um resultado aceitável vai depender do tipo de movimento desejado para a plataforma de Stewart e dos erros aceitáveis estipulados para o processo iterativo.

Para um erro total da soma de $F1, F2$ e $F3$ inferior a 0.0001, ou a soma de $\Delta X[1], \Delta X[2]$ e $\Delta X[3]$ inferior a 0.00001 em um movimento senoidal planar, o método numérico leva 4 a 9 iterações para convergir para o resultado.

Uma vez conhecidos os pontos da plataforma superior, deve-se então determinar a posição do sistema de referência móvel em relação ao sistema de referência inercial, afinal, o movimento da plataforma superior é dado em função do movimento deste referencial e não dos pontos de seus vértices. Porém, esta é uma tarefa fácil, uma vez que a plataforma superior é um triângulo equilátero. A posição do sistema de coordenadas centrado no centro de massa da plataforma superior é dada pela média das coordenadas X, Y , e Z de cada um dos pontos dos vértices.

$$p_x = \frac{1}{3} \sum_{i=1}^3 X_{Ti} \quad (75)$$

$$p_y = \frac{1}{3} \sum_{i=1}^3 Y_{Ti} \quad (76)$$

$$p_z = \frac{1}{3} \sum_{i=1}^3 Z_{Ti} \quad (77)$$

Definida a origem do sistema de coordenadas, resta definir a orientação. Conhecidos três pontos, determina-se um plano com relação ao referencial inercial. Conhecido o plano, determina-se sua orientação.

$$\gamma = \text{Arctan2} \left\{ \frac{Y_{T1} - 2Y_{T2} + Y_{T3}}{\sqrt{3}(Y_{T1} - Y_{T3})} \right\} \quad (78)$$

$$\alpha = \text{Arctan2} \left\{ -\frac{2[X_{T1}(Z_{T3} - Z_{T2}) + X_{T2}(Z_{T1} - Z_{T3}) - X_{T3}(Z_{T1} - Z_{T2})]}{a(Y_{T1} - 2Y_{T2} + Y_{T3}) \sin \gamma + \sqrt{3}a(Y_{T1} - Y_{T3}) \cos \gamma} \right\} \quad (79)$$

$$\beta = \text{Arctan2} \left\{ -\frac{(Z_{T1} - 2Z_{T2} + Z_{T3}) \cos \gamma + \sqrt{3}(Z_{T1} - Z_{T3}) \sin \gamma}{(X_{T1} - 2X_{T2} + X_{T3}) \cos \gamma - \sqrt{3}(X_{T1} - X_{T3}) \sin \gamma} \right\} \quad (80)$$

Em que γ é o ângulo de rotação do sistema de coordenadas móvel em torno do eixo Z do sistema de coordenadas inercial, α em torno do eixo X e β do eixo Y. Utilizando a função “Arctan2”, o valor do ângulo encontrado será sempre positivo, evitando inconsistências.

$$\text{Arctan2} \left\{ \frac{N}{D} \right\} = \text{Arctan} \left\{ \frac{N}{D} \right\} \quad \text{if } D > 0 \quad (81)$$

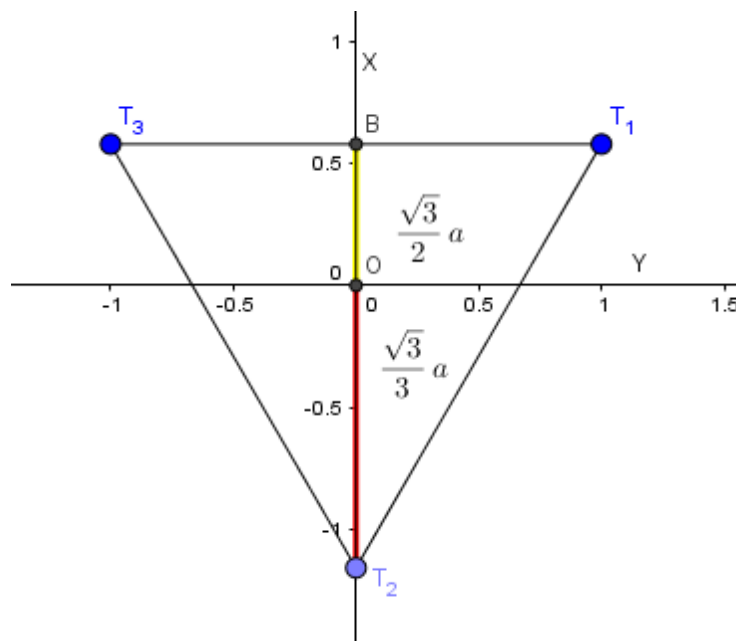
$$\text{Arctan2} \left\{ \frac{N}{D} \right\} = \text{Arctan} \left\{ \frac{N}{D} \right\} - \text{Sinal}(N) \times 180 \quad \text{if } D < 0 \quad (82)$$

APÊNDICE 2 - Solução de cinemática inversa de uma plataforma de Stewart

Uma vez determinado o movimento desejado para a Plataforma de Stewart, deve-se determinar qual o comprimento de cada atuador em cada instante de tempo de forma que, combinados, resultem no movimento desejado da plataforma superior.

Sendo assim, partindo do princípio que a posição e a orientação do sistema de coordenadas móvel em relação ao sistema de coordenadas inercial são conhecidas, pode-se determinar as coordenadas x , y e z das extremidades da plataforma superior, e posteriormente, utilizando uma transformada, determinar a posição destes pontos em relação ao sistema de coordenadas inercial.

Figura 78 - Plataforma superior.



Fonte: elaborada pelo autor.

Conforme visto na Figura 78, o segmento $\overline{T_2O}$ representa a distância do ponto T_2 ao centro de massa do triângulo. Por outro lado, o segmento $\overline{T_2B}$

representa a altura do triângulo. Sendo assim, a coordenada X dos pontos T_1 e T_3 nada mais é que a altura menos a posição do centro de massa do triângulo. Em T_2 a coordenada X pode ser obtida através da equação que define a posição do centro de massa. Lembrando que o triângulo tem aresta de comprimento “a”, as coordenadas x,y,z dos pontos T_1, T_2 e T_3 , em relação ao sistema de coordenadas móvel, são:

$$x_{T1} = \frac{\sqrt{3}}{2}a - \frac{\sqrt{3}}{3}a = \frac{\sqrt{3}}{6}a \quad (83)$$

$$y_{T1} = \frac{1}{2}a \quad (84)$$

$$z_{T1} = 0 \quad (85)$$

$$x_{T2} = \frac{\sqrt{3}}{3}a \quad (86)$$

$$y_{T2} = 0 \quad (87)$$

$$z_{T2} = 0 \quad (88)$$

$$x_{T3} = x_{T1} = \frac{\sqrt{3}}{6}a \quad (89)$$

$$y_{T3} = -\frac{1}{2}a \quad (90)$$

$$z_{T3} = 0 \quad (91)$$

Utilizando uma matriz de transformação homogênea de Tait-Bryan YXZ, conforme a que segue:

$$[Y_1X_2Z_3] = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 & p_x \\ c_2s_3 & c_2c_3 & -s_2 & p_y \\ c_1s_2s_3 - c_3s_1 & c_1c_3s_2 + s_1s_3 & c_1c_2 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (92)$$

Pode-se determinar as coordenadas dos pontos desejados.

$$\begin{bmatrix} X_{T1} \\ Y_{T1} \\ Z_{T1} \\ 1 \end{bmatrix} = [Y_1X_2Z_3] \begin{bmatrix} x_{T1} \\ y_{T1} \\ z_{T1} \\ 1 \end{bmatrix} \quad (93)$$

$$\begin{bmatrix} X_{T2} \\ Y_{T2} \\ Z_{T2} \\ 1 \end{bmatrix} = [Y_1 X_2 Z_3] \begin{bmatrix} x_{T2} \\ y_{T2} \\ z_{T2} \\ 1 \end{bmatrix} \quad (94)$$

$$\begin{bmatrix} X_{T3} \\ Y_{T3} \\ Z_{T3} \\ 1 \end{bmatrix} = [Y_1 X_2 Z_3] \begin{bmatrix} x_{T3} \\ y_{T3} \\ z_{T3} \\ 1 \end{bmatrix} \quad (95)$$

O que resultará em:

$$X_{T1} = px + \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma + 60) + \cos(\beta) \cos(\gamma + 60)] \quad (96)$$

$$Y_{T1} = py + \frac{a}{\sqrt{3}} [\cos(\alpha) \sin(\gamma + 60)] \quad (97)$$

$$Z_{T1} = pz + \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma + 60) - \sin(\beta) \cos(\gamma + 60)] \quad (98)$$

$$X_{T2} = px - \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma) + \cos(\beta) \cos(\gamma)] \quad (99)$$

$$Y_{T2} = py - \frac{a}{\sqrt{3}} [\cos(\alpha) \sin(\gamma)] \quad (100)$$

$$Z_{T2} = pz - \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma - 60) + \cos(\beta) \cos(\gamma)] \quad (101)$$

$$X_{T3} = px + \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma - 60) + \cos(\beta) \cos(\gamma - 60)] \quad (102)$$

$$Y_{T3} = py + \frac{a}{\sqrt{3}} [\cos(\alpha) \sin(\gamma - 60)] \quad (103)$$

$$Z_{T3} = pz + \frac{a}{\sqrt{3}} [\sin(\alpha) \sin(\beta) \sin(\gamma - 60) - \sin(\beta) \cos(\gamma - 60)] \quad (104)$$

O “60°” nas equações resultantes não tem significado físico. Surge a partir da simplificação dos termos $\frac{\sqrt{3}}{6}a$ e $\frac{\sqrt{3}}{3}a$ para que fiquem na forma $\frac{1}{\sqrt{3}}a$. A forma simplificada é estética, visa reduzir a forma da equação e apresentar o resultado de forma mais amigável, mas desde que o produto das matrizes não apresente erros, não há necessidade de simplificação. Para chegar aos resultados como apresentados, faz-se uso das leis trigonométricas de cosseno de arco duplo e seno de arco duplo. Para melhor compreensão, a obtenção de X_{T1} será apresentada:

$$\begin{bmatrix} X_{T1} \\ Y_{T1} \\ Z_{T1} \\ 1 \end{bmatrix} = [Y_1 X_2 Z_3] \begin{bmatrix} x_{T1} \\ y_{T1} \\ z_{T1} \\ 1 \end{bmatrix} \quad (105)$$

$$X_{T1} = x_{T1}(c_1 c_3 + s_1 s_2 s_3) + y_{T1}(c_3 s_1 s_2 - c_1 s_3) + z_{T1}(c_2 s_1) + px \quad (106)$$

$$X_{T1} = \frac{\sqrt{3}}{6}a(c_1 c_3 + s_1 s_2 s_3) + \frac{1}{2}a(c_3 s_1 s_2 - c_1 s_3) + 0(c_2 s_1) + px \quad (107)$$

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[\frac{1}{2}(c_1c_3 + s_1s_2s_3) + \frac{\sqrt{3}}{2}(c_3s_1s_2 - c_1s_3)] \quad (108)$$

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[c(60)(c_1c_3 + s_1s_2s_3) + s(60)(c_3s_1s_2 - c_1s_3)] \quad (109)$$

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[c(60)c_1c_3 + c(60)s_1s_2s_3 + s(60)c_3s_1s_2 - s(60)c_1s_3] \quad (110)$$

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[s_1s_2(s_3c(60) + c_3s(60)) + c_1(c_3c(60) - s_3s(60))] \quad (111)$$

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[s_1s_2(s_{3+60}) + c_1(c_{3+60})] \quad (112)$$

Sendo que 1 é equivalente à rotação β em torno do eixo Y, 2 equivale à rotação α em torno do eixo X e 3 à rotação γ em torno do eixo Z. Escrevendo a Equação (112) nesta forma, obtém-se:

$$X_{T1} = px + \frac{1}{\sqrt{3}}a[\text{sen}(\beta)\text{sen}(\alpha)\text{sen}(\gamma + 60) + \cos(\beta)\cos(\gamma + 60)] \quad (113)$$

A obtenção das demais coordenadas é feita de forma similar à esta.

Com as coordenadas X, Y, Z conhecidas, resta determinar as dimensões dos atuadores. Como os pontos das duas extremidades do atuador são conhecidos (por exemplo, para o atuador L1, conhece-se o ponto B1 e T1), pode-se então criar um vetor com estes dois pontos e, tirando o módulo deste vetor, determinar o comprimento de L1:

$$L_1 = \sqrt{(X_{T1} - X_{B1})^2 + (Y_{T1} - Y_{B1})^2 + (Z_{T1} - Z_{B1})^2} \quad (114)$$

$$L_1 = \sqrt{\left(X_{T1} - \frac{d}{2\sqrt{3}} - \frac{db}{\sqrt{3}}\right)^2 + \left(Y_{T1} - \frac{d}{2}\right)^2 + Z_{T1}^2} \quad (115)$$

De forma similar, pode-se determinar o comprimento dos demais atuadores:

$$L_2 = \sqrt{\left(X_{T1} - \frac{d}{2\sqrt{3}} + \frac{b}{2\sqrt{3}}\right)^2 + \left(Y_{T1} - \frac{d}{2} - \frac{b}{2}\right)^2 + Z_{T1}^2} \quad (116)$$

$$L_3 = \sqrt{\left(X_{T2} - \frac{d}{\sqrt{3}} - \frac{b}{2\sqrt{3}}\right)^2 + \left(Y_{T2} - \frac{b}{2}\right)^2 + Z_{T2}^2} \quad (117)$$

$$L_4 = \sqrt{\left(X_{T2} - \frac{d}{\sqrt{3}} - \frac{b}{2\sqrt{3}}\right)^2 + \left(Y_{T2} + \frac{b}{2}\right)^2 + Z_{T2}^2} \quad (118)$$

$$L_5 = \sqrt{\left(X_{T3} - \frac{d}{2\sqrt{3}} + \frac{b}{2\sqrt{3}}\right)^2 + \left(Y_{T3} + \frac{d}{2} + \frac{b}{2}\right)^2 + Z_{T3}^2} \quad (119)$$

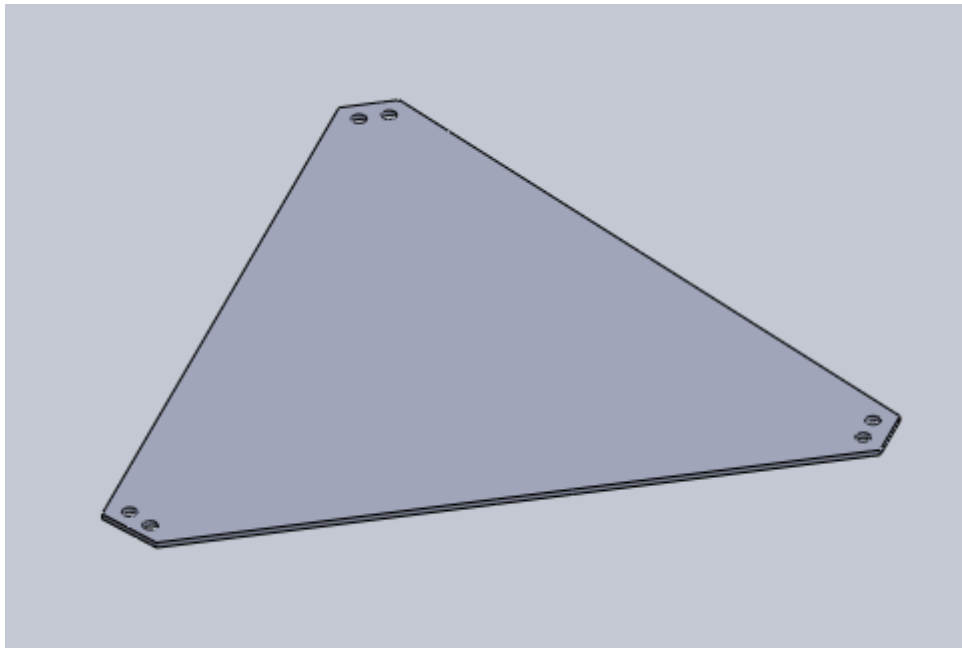
$$L_6 = \sqrt{\left(X_{T3} - \frac{d}{2\sqrt{3}} - \frac{b}{\sqrt{3}}\right)^2 + \left(Y_{T3} + \frac{d}{2}\right)^2 + Z_{T3}^2} \quad (120)$$

Sendo estes os comprimentos que, neste instante, farão com que a plataforma superior fique na posição desejada.

APÊNDICE 3 – Componentes para construção do modelo simplificado.

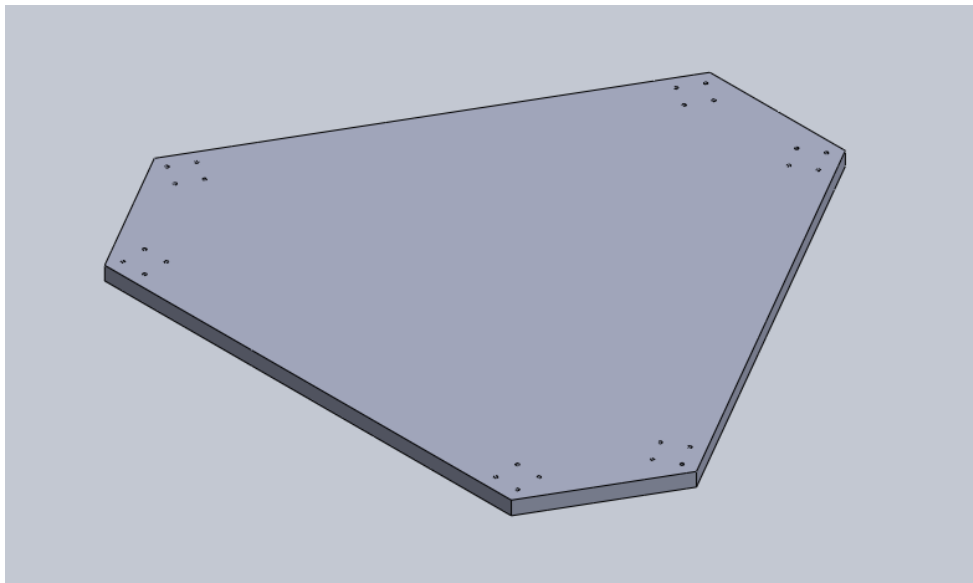
Antes da construção efetiva do modelo, construiu-se um projeto em *SolidWorks*. As peças dos kit's LEGO foram obtidas nos links das referências [27] e [28], que contém um conjunto de vários elementos, dentre eles, os aqui utilizados. Os componentes da empresa Robotics foram obtidos no link da referência [26]. Para a construção do modelo simplificado foram utilizadas placas de acrílico leitoso de 2mm de espessura para as plataformas superior e inferior, conforme as Figuras 79 e 80, respectivamente.

Figura 79 - Plataforma superior.



Fonte: componente em SolidWorks.

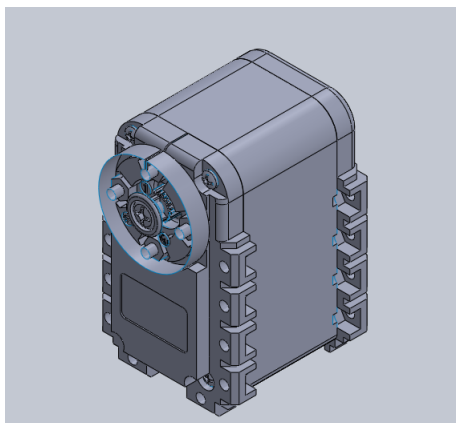
Figura 80 - Plataforma inferior.



Fonte: componente em SolidWorks.

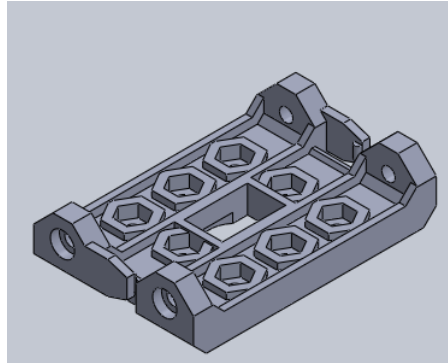
Como atuador, utilizou-se motores AX-12, conforme o da Figura 81, produzidos pela empresa Robotics. Para uni-los à plataforma inferior, utilizou-se o conector FP04-F3, também da Robotics, conforme a Figura 82.

Figura 81 - AX12.



Fonte: componente em SolidWorks.

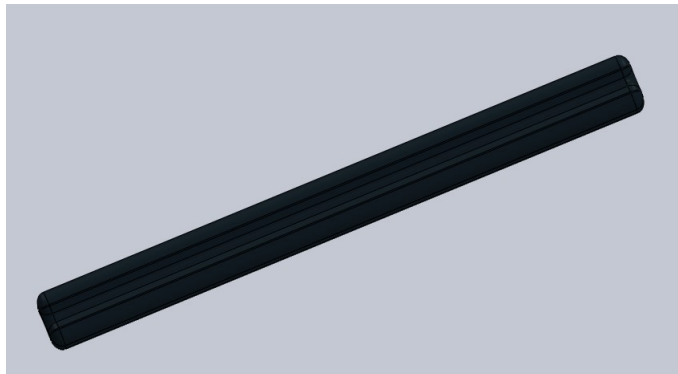
Figura 82 - Conector FP04-F3.



Fonte: componente em SolidWorks.

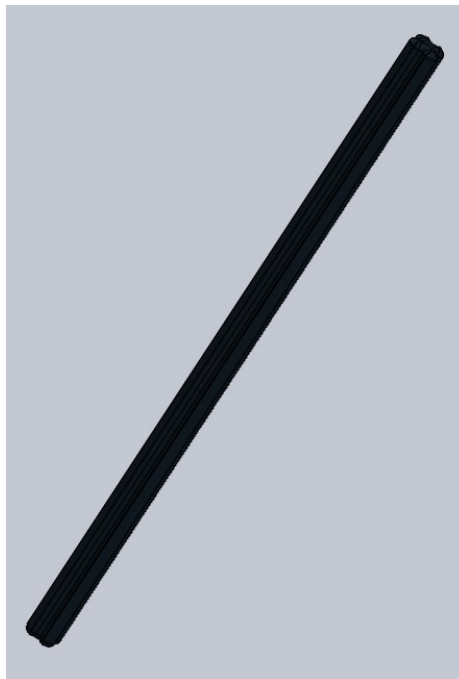
Os atuadores foram acoplados à um sistema de biela-manivela, de forma que todo o conjunto trabalhe como um atuador linear. Para construir o sistema de biela-manivela foram utilizadas peças do kit *Mindstorm* produzidas pela LEGO. O conjunto é composto por dois eixos, *Technic Axle 6 (black)*, como o da Figura 83, e *Technic Axle 12 (black)*, conforme Figura 84 e duas juntas universais, conforme a Figura 85.

Figura 83 - Technic axle 6 (black).



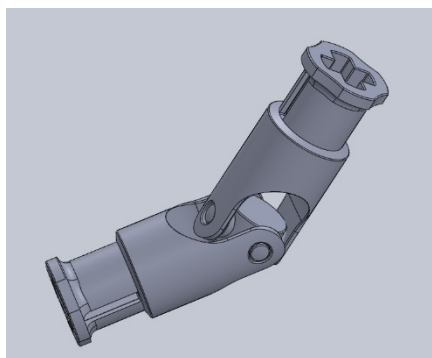
Fonte: componente em SolidWorks.

Figura 84 - Technic axle 12 (black).



Fonte: componente em SolidWorks.

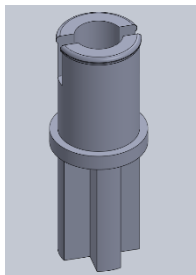
Figura 85 - Junta universal.



Fonte: componente em SolidWorks.

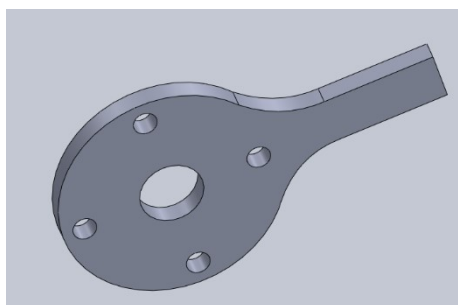
Um conector eixo para pino, chamado *axle to pin* 43093, que pode ser visto na Figura 86, e um elemento produzido em impressora 3D que conecta o eixo de giro do motor ao restante dos elementos, conforme Figura 87, conectados por um conector eixo para eixo, chamado *cross axle extension* 2M, como o da Figura 88, completam o mecanismo.

Figura 86 - Axle to pin 43093.



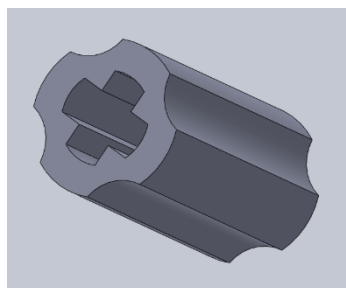
Fonte: componente em SolidWorks.

Figura 87 - Conector do eixo do motor.



Fonte: componente em SolidWorks.

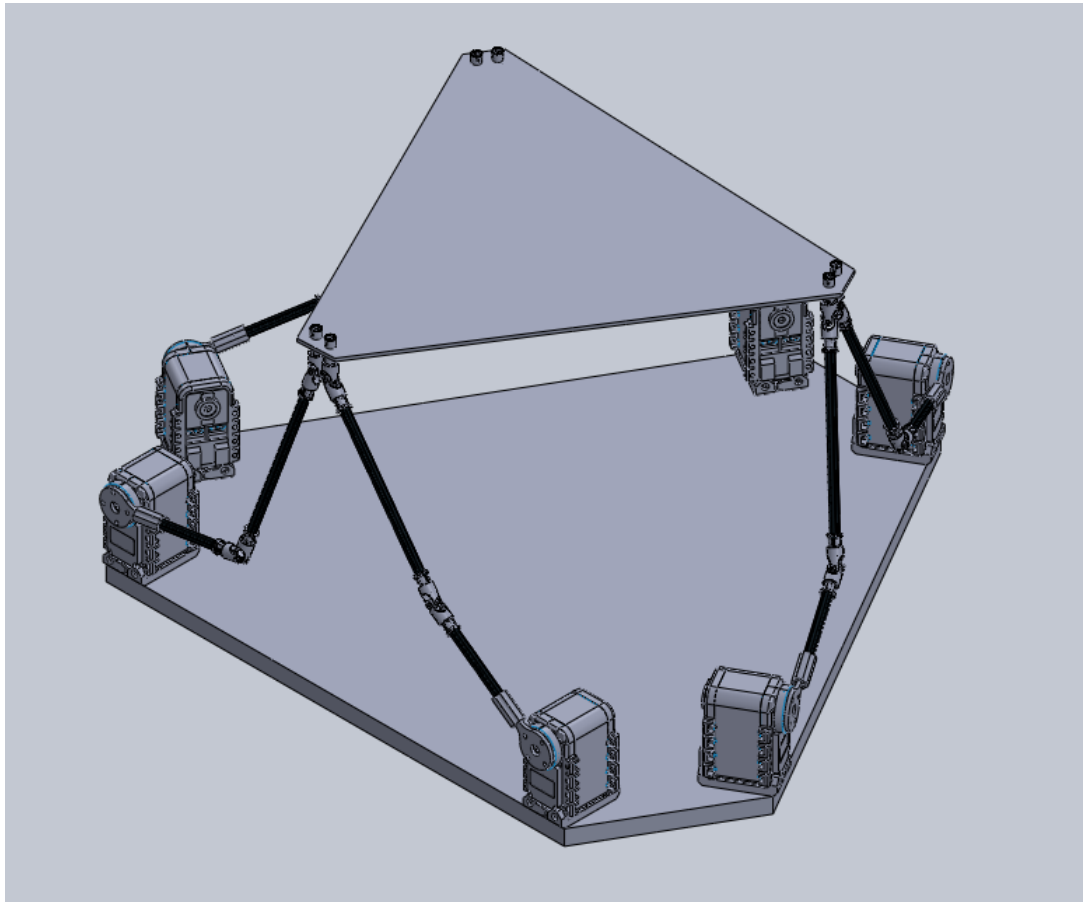
Figura 88 - Cross axle extention 2M.



Fonte: componente em SolidWorks.

A montagem completa da plataforma com todos os componentes citados, pode ser vista na Figura 89.

Figura 89 - Desenho da montagem.



Fonte: componente em SolidWorks.

APÊNDICE 4 – Software em MATLAB para simulação de modelos do protótipo.

Cinemática Direta (FK – Forward Kinematics)

```
function [px,py,pz,alpha,beta,gamma] = FK(a,b,d,L1,L2,L3,L4,L5,L6)
```

```
% Forward Kinematics
```

```
% a = sym('a');
```

```
% Xt1 = sym('Xt1');Xt2 = sym('Xt2');Xt3 = sym('Xt3');
```

```
% Xp1 = sym('Xp1');Xp2 = sym('Xp2');Xp3 = sym('Xp3');
```

```
% Yp1 = sym('Yp1');Yp2 = sym('Yp2');Yp3 = sym('Yp3');
```

```
% h1 = sym('h1');h2 = sym('h2');h3 = sym('h3');
```

```
ntrial = 1000;
```

```
errtol_F = 1e-4;
```

```
errtol_X = 1e-5;
```

```
err_X = 1;
```

```
err_F = 1;
```

```
n = 0;
```

```
P1 = 1/(2*b)*(b*b + L1*L1 - L2*L2);
```

```
P2 = 1/(2*b)*(b*b + L3*L3 - L4*L4);
```

```
P3 = 1/(2*b)*(b*b + L5*L5 - L6*L6);
```

```
h1 = sqrt(L1*L1 - P1*P1);
```

```
h2 = sqrt(L3*L3 - P2*P2);
```

```
h3 = sqrt(L5*L5 - P3*P3);
```

```
Xp1 = sqrt(3)/6*(2*b + d - 3*P1);
```

```
Xp2 = - sqrt(3)/6*(b + 2*d);
```

```
Xp3 = - sqrt(3)/6*(b-d-3*P3);
```

```
Yp1 = 1/2*(d+P1);
```

```
Yp2 = 1/2*(b-2*P2);
```

```
Yp3 = - 1/2*(b+d-P3);
```

```
Xt1 = Xp1; Xt2 = Xp2; Xt3 = Xp3;
```

```
while err_F > errtol_F && err_X > errtol_X && n < ntrial
```

```
F1 = a*a + 2*Xt1*Xt2 - 2*Xt1*(Xp1 + sqrt(3)*(Yp1 - Yp2)) - 2*Xp2*Xt2 - ((sqrt(3)*Xp1 - Yp1 + Yp2)*(sqrt(3)*Xp1 - Yp1 + Yp2) + (h1*h1 + h2*h2) - 4*Xp1*Xp1 - Xp2*Xp2) + 2*sqrt((h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1))*(h2*h2 - (Xt2 - Xp2)*(Xt2 - Xp2)));
```

```
F2 = a*a - 4*Xt1*Xt3 - 2*Xt1*(Xp1 - 3*Xp3 + sqrt(3)*(Yp1 - Yp3)) - 2*Xt3*(-3*Xp1 + Xp3 + sqrt(3)*(Yp1-Yp3)) - ((sqrt(3)*(Xp1 + Xp3) - Yp1 + Yp3)*(sqrt(3)*(Xp1 + Xp3) - Yp1 + Yp3) + (h1*h1 + h3*h3) - 4*Xp1*Xp1 - 4*Xp3*Xp3) + 2*sqrt((h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1))*(h3*h3 - 4*(Xt3 - Xp3)*(Xt3 - Xp3)));
```

```
F3 = a*a + 2*Xt2*Xt3 - 2*Xt3*(Xp3 + sqrt(3)*(Yp2 - Yp3)) - 2*Xp2*Xt2 - ((sqrt(3)*Xp3 - Yp2 + Yp3)*(sqrt(3)*Xp3 - Yp2 + Yp3) + (h2*h2 + h3*h3) - 4*Xp3*Xp3 - Xp2*Xp2) + 2*sqrt((h3*h3 - 4*(Xt3-Xp3)*(Xt3-Xp3))*(h2*h2 - (Xt2 - Xp2)*(Xt2 - Xp2)));
```

```
%J11 = diff(F1,Xt1);
```

```
J11 = 2*Xt2 - 2*Xp1 - 2*3^(1/2)*(Yp1 - Yp2) + ((h2^2 - (Xp2 - Xt2)^2)*(8*Xp1 - 8*Xt1))/((h1^2 - 4*(Xp1 - Xt1)^2)*(h2^2 - (Xp2 - Xt2)^2))^(1/2);
```

```
%J12 = diff(F1,Xt2);
```

```
J12 = 2*Xt1 - 2*Xp2 + ((h1^2 - 4*(Xp1 - Xt1)^2)*(2*Xp2 - 2*Xt2))/((h1^2 - 4*(Xp1 - Xt1)^2)*(h2^2 - (Xp2 - Xt2)^2))^(1/2);
```

```
%J13 = diff(F1,Xt3);
```

```
J13 = 0;
```

```
%J21 = diff(F2,Xt1);
```

```
J21 = 6*Xp3 - 2*Xp1 - 4*Xt3 - 2*3^(1/2)*(Yp1 - Yp3) + ((h3^2 - 4*(Xp3 - Xt3)^2)*(8*Xp1 - 8*Xt1))/((h1^2 - 4*(Xp1 - Xt1)^2)*(h3^2 - 4*(Xp3 - Xt3)^2))^(1/2);
```

```
%J22 = diff(F2,Xt2);
```

```
J22 = 0;
```

```
%J23 = diff(F2,Xt3);
```

```
J23 = 6*Xp1 - 2*Xp3 - 4*Xt1 - 2*3^(1/2)*(Yp1 - Yp3) + ((h1^2 - 4*(Xp1 - Xt1)^2)*(8*Xp3 - 8*Xt3))/((h1^2 - 4*(Xp1 - Xt1)^2)*(h3^2 - 4*(Xp3 - Xt3)^2))^(1/2);
```

```
%J31 = diff(F3,Xt1);
```

```
J31 = 0;
```

```
%J32 = diff(F3,Xt2);
```

$$J32 = 2*Xt3 - 2*Xp2 + ((h3^2 - 4*(Xp3 - Xt3)^2)*(2*Xp2 - 2*Xt2))/((h2^2 - (Xp2 - Xt2)^2)*(h3^2 - 4*(Xp3 - Xt3)^2))^{(1/2)};$$

$$\%J33 = \text{diff}(F3, Xt3);$$

$$J33 = 2*Xt2 - 2*Xp3 - 2*3^{(1/2)}*(Yp2 - Yp3) + ((h2^2 - (Xp2 - Xt2)^2)*(8*Xp3 - 8*Xt3))/((h2^2 - (Xp2 - Xt2)^2)*(h3^2 - 4*(Xp3 - Xt3)^2))^{(1/2)};$$

$$J = [J11 \ J12 \ J13 ; J21 \ J22 \ J23; J31 \ J32 \ J33];$$

$$F = [F1 ; F2 ; F3];$$

$$\text{deltaX} = \text{inv}(J)*(-F);$$

$$Xt1 = Xt1 + \text{deltaX}(1);$$

$$Xt2 = Xt2 + \text{deltaX}(2);$$

$$Xt3 = Xt3 + \text{deltaX}(3);$$

$$\text{err_X} = \text{abs}(\text{deltaX}(1)) + \text{abs}(\text{deltaX}(2)) + \text{abs}(\text{deltaX}(3));$$

$$\text{err_F} = \text{abs}(F1) + \text{abs}(F2) + \text{abs}(F3);$$

$$n = n + 1;$$

end

$$Yt1 = \sqrt{3}*Xt1 - (\sqrt{3}*Xp1 - Yp1);$$

$$Yt2 = Yp2;$$

$$Yt3 = -\sqrt{3}*Xt3 + (\sqrt{3}*Xp3 + Yp3);$$

$$Zt1 = \sqrt{h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1)};$$

$$Zt2 = \sqrt{h2*h2 - (Xt2-Xp2)*(Xt2-Xp2)};$$

$$Zt3 = \sqrt{h3*h3 - 4*(Xt3-Xp3)*(Xt3-Xp3)};$$

$$p_x = (X_{t1} + X_{t2} + X_{t3})/3;$$

$$p_y = (Y_{t1} + Y_{t2} + Y_{t3})/3;$$

$$p_z = (Z_{t1} + Z_{t2} + Z_{t3})/3;$$

$$\gamma = \text{atan2}((Y_{t1} - 2*Y_{t2} + Y_{t3}), (\sqrt{3}*(Y_{t1} - Y_{t3})));$$

$$\alpha = \text{atan2}((-2*(X_{t1}*(Z_{t3} - Z_{t2}) + X_{t2}*(Z_{t1} - Z_{t3}) - X_{t3}*(Z_{t1} - Z_{t2}))), (a*(Y_{t1} - 2*Y_{t2} + Y_{t3})*\sin(\gamma) + \sqrt{3}*a*(Y_{t1} - Y_{t3})*\cos(\gamma)));$$

$$\beta = \text{atan2}((- (Z_{t1} - 2*Z_{t2} + Z_{t3})*\cos(\gamma) + \sqrt{3}*(Z_{t1} - Z_{t3})*\sin(\gamma)), ((X_{t1} - 2*X_{t2} + X_{t3})*\cos(\gamma) - \sqrt{3}*(X_{t1} - X_{t3})*\sin(\gamma)));$$

end

Cinemática Inversa (IK – Inverse Kinematics)

function [L1,L2,L3,L4,L5,L6,Xt1,Xt2,Xt3,Yt1,Yt2,Yt3,Zt1,Zt2,Zt3] =
IK(a,b,d,px,py,pz,alpha,beta,gamma)

%Inverse Kinematics

$$X_{t1} = p_x + a/\sqrt{3}*(\sin(\alpha)*\sin(\beta)*\sin(\gamma + \pi/3) + \cos(\beta)*\cos(\gamma + \pi/3));$$

$$X_{t2} = p_x - a/\sqrt{3}*(\sin(\alpha)*\sin(\beta)*\sin(\gamma) + \cos(\beta)*\cos(\gamma));$$

$$X_{t3} = p_x + a/\sqrt{3}*(\sin(\alpha)*\sin(\beta)*\sin(\gamma - \pi/3) + \cos(\beta)*\cos(\gamma - \pi/3));$$

$$Y_{t1} = p_y + a/\sqrt{3}*(\cos(\alpha)*\sin(\gamma + \pi/3));$$

$$Y_{t2} = p_y - a/\sqrt{3}*(\cos(\alpha)*\sin(\gamma));$$

$$Y_{t3} = p_y + a/\sqrt{3}*(\cos(\alpha)*\sin(\gamma - \pi/3));$$

$$Z_{t1} = p_z + a/\sqrt{3}*(\sin(\alpha)*\cos(\beta)*\sin(\gamma + \pi/3) - \sin(\beta)*\cos(\gamma + \pi/3));$$

$Zt2 = pz - a/\sqrt{3} * (\sin(\alpha) * \cos(\beta) * \sin(\gamma) - \sin(\beta) * \cos(\gamma));$

$Zt3 = pz + a/\sqrt{3} * (\sin(\alpha) * \cos(\beta) * \sin(\gamma - \pi/3) - \sin(\beta) * \cos(\gamma - \pi/3));$

$L1 = \sqrt{(Xt1 - d/(2*\sqrt{3})) - b/\sqrt{3}} * (Xt1 - d/(2*\sqrt{3})) - b/\sqrt{3} + (Yt1 - d/2) * (Yt1 - d/2) + Zt1 * Zt1);$

$L2 = \sqrt{(Xt1 - d/(2*\sqrt{3})) + b/(2*\sqrt{3}))} * (Xt1 - d/(2*\sqrt{3})) + b/(2*\sqrt{3}) + (Yt1 - d/2 - b/2) * (Yt1 - d/2 - b/2) + Zt1 * Zt1);$

$L3 = \sqrt{(Xt2 + d/(\sqrt{3})) + b/(2*\sqrt{3}))} * (Xt2 + d/(\sqrt{3})) + b/(2*\sqrt{3}) + (Yt2 - b/2) * (Yt2 - b/2) + Zt2 * Zt2);$

$L4 = \sqrt{(Xt2 + d/(\sqrt{3})) + b/(2*\sqrt{3}))} * (Xt2 + d/(\sqrt{3})) + b/(2*\sqrt{3}) + (Yt2 + b/2) * (Yt2 + b/2) + Zt2 * Zt2);$

$L5 = \sqrt{(Xt3 - d/(2*\sqrt{3})) + b/(2*\sqrt{3}))} * (Xt3 - d/(2*\sqrt{3})) + b/(2*\sqrt{3}) + (Yt3 + b/2 + d/2) * (Yt3 + b/2 + d/2) + Zt3 * Zt3);$

$L6 = \sqrt{(Xt3 - d/(2*\sqrt{3})) - b/\sqrt{3}} * (Xt3 - d/(2*\sqrt{3})) - b/\sqrt{3} + (Yt3 + d/2) * (Yt3 + d/2) + Zt3 * Zt3);$

end

Geração de trajetória (MM – Movement Maker)

clc; clear; close all;

%Platform Dimensions

a = 10;

b = 15;

d = 1;

Ma = 4;

Ba = 10;

```

Xb1 = sqrt(3)/6*(2*b + d);
Xb2 = - sqrt(3)/6*(b - d);
Xb3 = - sqrt(3)/6*(b + 2*d);
Xb4 = - sqrt(3)/6*(b + 2*d);
Xb5 = - sqrt(3)/6*(b - d);
Xb6 = sqrt(3)/6*(2*b + d);

```

```

Yb1 = 1/2*d;
Yb2 = 1/2*(b + d);
Yb3 = 1/2*b;
Yb4 = - 1/2*b;
Yb5 = - 1/2*(b + d);
Yb6 = - 1/2*d;

```

```

Zb1 = 0;
Zb2 = 0;
Zb3 = 0;
Zb4 = 0;
Zb5 = 0;
Zb6 = 0;

```

```

t = 0;
while 1

```

```

px = 3*sin(t*pi/100);
py = 0; %2*sin(t*pi/100);

```

pz = 8; %5 + 1*cos(t*pi/100);

alpha = 0; %pi/12*sin(t*pi/100);

beta = 0;

gamma = 0; %pi/12*sin(t*pi/100);

[L1,L2,L3,L4,L5,L6,Xt1,Xt2,Xt3,Yt1,Yt2,Yt3,Zt1,Zt2,Zt3] =
IK(a,b,d,px,py,pz,alpha,beta,gamma);

[px1,py1,pz1,alpha1,beta1,gamma1] = FK(a,b,d,L1,L2,L3,L4,L5,L6);

P1 = 1/(2*b)*(b*b + L1*L1 - L2*L2);

P2 = 1/(2*b)*(b*b + L3*L3 - L4*L4);

P3 = 1/(2*b)*(b*b + L5*L5 - L6*L6);

h1 = sqrt(L1*L1 - P1*P1);

h2 = sqrt(L3*L3 - P2*P2);

h3 = sqrt(L5*L5 - P3*P3);

thetaL1 = asin(h1/L1) - (acos((Ma*Ma + L1*L1 - Ba*Ba)/(2*Ma*L1)));

MaL1 = abs(Ma*cos(thetaL1));

L1x = Xb1 - sqrt(3)/2*MaL1;

L1y = Yb1 + MaL1*1/2;

L1z = Ma*sin(thetaL1);

thetaL2 = asin(h1/L2) - (acos((Ma*Ma + L2*L2 - Ba*Ba)/(2*Ma*L2)));

MaL2 = abs(Ma*cos(thetaL2));

L2x = Xb2 + sqrt(3)/2*MaL2;

L2y = Yb2 - MaL2*1/2;

$$L2z = Ma \cdot \sin(\theta_{L2});$$

$$\theta_{L3} = \arcsin(h2/L3) - (\arccos((Ma \cdot Ma + L3 \cdot L3 - Ba \cdot Ba)/(2 \cdot Ma \cdot L3)));$$

$$MaL3 = \text{abs}(Ma \cdot \cos(\theta_{L3}));$$

$$L3x = Xb3;$$

$$L3y = Yb3 - MaL3;$$

$$L3z = Ma \cdot \sin(\theta_{L3});$$

$$\theta_{L4} = \arcsin(h2/L4) - (\arccos((Ma \cdot Ma + L4 \cdot L4 - Ba \cdot Ba)/(2 \cdot Ma \cdot L4)));$$

$$MaL4 = \text{abs}(Ma \cdot \cos(\theta_{L4}));$$

$$L4x = Xb4;$$

$$L4y = Yb4 + MaL4;$$

$$L4z = Ma \cdot \sin(\theta_{L4});$$

$$\theta_{L5} = \arcsin(h3/L5) - (\arccos((Ma \cdot Ma + L5 \cdot L5 - Ba \cdot Ba)/(2 \cdot Ma \cdot L5)));$$

$$MaL5 = \text{abs}(Ma \cdot \cos(\theta_{L5}));$$

$$L5x = Xb5 + \sqrt{3}/2 \cdot MaL5;$$

$$L5y = Yb5 + MaL5 \cdot 1/2;$$

$$L5z = Ma \cdot \sin(\theta_{L5});$$

$$\theta_{L6} = \arcsin(h3/L6) - (\arccos((Ma \cdot Ma + L6 \cdot L6 - Ba \cdot Ba)/(2 \cdot Ma \cdot L6)));$$

$$MaL6 = \text{abs}(Ma \cdot \cos(\theta_{L6}));$$

$$L6x = Xb6 - \sqrt{3}/2 \cdot MaL6;$$

$$L6y = Yb6 - MaL6 \cdot 1/2;$$

$$L6z = Ma \cdot \sin(\theta_{L6});$$

figure(1)

```
X = [ Xb1 Xt1 Xb2 Xb3 Xt2 Xb4 Xb5 Xt3 Xb6 Xb1 Xb2 Xb3 Xb4 Xb5 Xb6 Xb1 Xt1 Xt2
Xt3 Xt1];
```

```
Y = [ Yb1 Yt1 Yb2 Yb3 Yt2 Yb4 Yb5 Yt3 Yb6 Yb1 Yb2 Yb3 Yb4 Yb5 Yb6 Yb1 Yt1 Yt2
Yt3 Yt1];
```

```
Z = [ Zb1 Zt1 Zb2 Zb3 Zt2 Zb4 Zb5 Zt3 Zb6 Zb1 Zb2 Zb3 Zb4 Zb5 Zb6 Zb1 Zt1 Zt2
Zt3 Zt1];
```

```
plot3(X,Y,Z);
```

```
view([-3,-1,1]);
```

```
axis([-15 15 -15 15 0 8]);
```

```
figure(2)
```

```
X = [ Xb1 L1x Xt1 L2x Xb2 Xb3 L3x Xt2 L4x Xb4 Xb5 L5x Xt3 L6x Xb6 Xb1 Xb2 Xb3
Xb4 Xb5 Xb6 L6x Xt3 Xt2 Xt1 Xt3];
```

```
Y = [ Yb1 L1y Yt1 L2y Yb2 Yb3 L3y Yt2 L4y Yb4 Yb5 L5y Yt3 L6y Yb6 Yb1 Yb2 Yb3
Yb4 Yb5 Yb6 L6y Yt3 Yt2 Yt1 Yt3];
```

```
Z = [ Zb1 L1z Zt1 L2z Zb2 Zb3 L3z Zt2 L4z Zb4 Zb5 L5z Zt3 L6z Zb6 Zb1 Zb2 Zb3
Zb4 Zb5 Zb6 L6z Zt3 Zt2 Zt1 Zt3];
```

```
plot3(X,Y,Z);
```

```
view([-1,-1,1]);
```

```
axis([-15 15 -15 15 0 8]);
```

```
t = t + 1;
```

```
end
```

APÊNDICE 5 – Cmakelist

Declara a versão mínima do Cmake que deve estar instalada (Recomenda-se a utilização especificando a versão das funcionalidades mais novas que estiver sendo utilizadas no projeto. Neste caso, o YARP, por exemplo, requer ao menos a versão 2.8.7)

```
cmake_minimum_required(VERSION 2.8.7)
```

#Recomenda-se atribuir um nome ao projeto

```
project(Stewart_Platform)
```

#Algumas definições gerais para o compilador (CXXFLAGS)

```
add_definitions(-std=c++11 -Wall `xml2-config --cflags`)
```

```
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY
${CMAKE_CURRENT_SOURCE_DIR}/build/lib) #Pasta padrão para colocar os
arquivos .o, .a, .so
```

```
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY
${CMAKE_CURRENT_SOURCE_DIR}/build/lib) #Pasta padrão para colocar os
arquivos .o, .a, .so (ainda não se a
```

```
include_directories(${CMAKE_CURRENT_SOURCE_DIR})
```

```
include_directories (${CMAKE_SOURCE_DIR}) #incluir os cabeçalhos em relação a
esta pasta (ex: p/ incluir a Timer.hpp, usar #include <general/Timer.hpp>
```

```
add_executable(StewartPlatform StewartPlatform.cpp)
```

```
find_package(Armadillo REQUIRED)
```

```
include_directories(${ARMADILLO_INCLUDE_DIRS})
```

```
add_library(u2d funcmovu2d.cpp dynamixel.c dxi_hal.c)
```

```
add_library(channel Channel.cpp)
```

```
target_link_libraries(channel u2d)
```

```
target_link_libraries(StewartPlatform ${ARMADILLO_LIBRARIES} channel)
```

APÊNDICE 6 – Código em C e C++**Stewart Platform.cpp**

```
#include <string.h>
#include <stdio.h>
#include <stdint.h>
#include <math.h>
#include <armadillo>
#include <Channel.hpp>
```

```
using namespace arma;
```

```
//Dimensões
```

```
#define a 240
```

```
#define b 300
```

```
#define d 100
```

```
#define Ma 95 // manivela
```

```
#define Ba 125 // biela
```

```
#define pi 3.1415926536
```

```
#define rad2deg 180/pi
```

```
//Posição de memória
```

```
#define P_GOAL_POSITION_L 30
```

```
#define USB2DXL 0
```

```
#define ID0 100
```

```
#define ID1 101
```

```
#define ID2 102
```

```
#define ID3 103
```

```
#define ID4 104
```

```
#define ID5 105
```

```
void Fk(double*, double*, double*, double*, double*, double*,double*);
```

```
void lk(double*, double, double, double, double, double, double);
```

```
int main( int argc, const char *argv[] )
```

```
{
```

```
    double pos_resolution = 0.29;
```

```
    double ref = 512;
```

```
    Channel com_port(USB2DXL);
```

```
    com_port.init(USB2DXL,1,255);
```

```
    int t = 0;
```

```
    double px;
```

```
    double py;
```

```
    double pz;
```

```
    double alpha;
```

```
    double beta;
```

```
    double gamma;
```

```

double L[6];

uint16_t data[2];

data[0] = 450;

data[1] = 100;

com_port.writeData((uint8_t) ID0, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

com_port.writeData((uint8_t) ID1, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

com_port.writeData((uint8_t) ID2, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

com_port.writeData((uint8_t) ID3, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

com_port.writeData((uint8_t) ID4, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

com_port.writeData((uint8_t) ID5, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0] );

sleep(5);

double L1;

double L2;

double L3;

double L4;

double L5;

double L6;

while(1){

    if(t<50){

```

```

    px = 0;      // Modelo de translação no eixo x
    py = 0;      // Modelo de translação no eixo y
    pz = 110;    // Modelo de translação no eixo z
    alpha = 0;   // Modelo de rotação em torno do eixo x
    beta = pi/24*sin(t*pi/5); // Modelo de rotação em torno do
eixo y
    gamma = 0;   // Modelo de rotação em torno do eixo z
}

if(t>50 && t<100){
    px = 0;      // Modelo de translação no eixo x
    py = 0;      // Modelo de translação no eixo y
    pz = 110;    // Modelo de translação no eixo z
    alpha = pi/24*sin(t*pi/5); // Modelo de rotação em torno do
eixo x
    beta = 0;    // Modelo de rotação em torno do eixo y
    gamma = 0;   // Modelo de rotação em torno do eixo z
}

if(t>100 && t<150){
    px = 0;      // Modelo de translação no eixo x
    py = 0;      // Modelo de translação no eixo y
    pz = 110;    // Modelo de translação no eixo z
    alpha = 0;   // Modelo de rotação em torno do eixo x
    beta = 0;    // Modelo de rotação em torno do eixo y
    gamma = pi/24*sin(t*pi/5); // Modelo de rotação em torno do
eixo z

```

```
}
```

```
if(t>150 && t<200){
```

```
px = 0;           // Modelo de translação no eixo x
```

```
py = 0;           // Modelo de translação no eixo y
```

```
pz = 110 + 20*sin(t*pi/5);           // Modelo de translação no eixo z
```

```
alpha = 0;        // Modelo de rotação em torno do eixo x
```

```
beta = 0;          // Modelo de rotação em torno do eixo y
```

```
gamma = 0;         // Modelo de rotação em torno do eixo z
```

```
}
```

```
if(t>200 && t<250){
```

```
px = 20*sin(t*pi/5);           // Modelo de translação no eixo x
```

```
py = 0;;           // Modelo de translação no eixo y
```

```
pz = 110;          // Modelo de translação no eixo z
```

```
alpha = 0;         // Modelo de rotação em torno do eixo x
```

```
beta = 0;          // Modelo de rotação em torno do eixo y
```

```
gamma = 0;         // Modelo de rotação em torno do eixo z
```

```
}
```

```
if(t>250 && t<300){
```

```
px = 0;           // Modelo de translação no eixo x
```

```
py = 20*sin(t*pi/5);           // Modelo de translação no eixo y
```

```
pz = 110;          // Modelo de translação no eixo z
```

```
alpha = 0;         // Modelo de rotação em torno do eixo x
```

```
beta = 0;          // Modelo de rotação em torno do eixo y
```

```

gamma = 0;          // Modelo de rotação em torno do eixo z
}

//~ printf("Points_Before px:%lf py:%lf pz:%lf ||| Angles_Before A:%lf
B:%lf G:%lf \n", px, py, pz, alpha, beta, gamma);

lk(&L[0],px,py,pz,alpha,beta,gamma);

Fk(&px,&py, &pz, &alpha, &beta, &gamma,&L[0]);

L1 = L[0];
L2 = L[1];
L3 = L[2];
L4 = L[3];
L5 = L[4];
L6 = L[5];

printf("Legs: L1:%lf L2:%lf L3:%lf L4:%lf L5:%lf L6:%lf \n",
L[0],L[1],L[2],L[3],L[4],L[5]);

//~ printf("Points_After px:%f py:%f pz:%f ||| Angles_After A:%f B:%f
G:%f \n", px, py, pz, alpha*rad2deg, beta*rad2deg, gamma*rad2deg);

float P1 = (b*b + L1*L1 - L2*L2)/(2*b);
float P2 = (b*b + L3*L3 - L4*L4)/(2*b);
float P3 = (b*b + L5*L5 - L6*L6)/(2*b);

float h1 = sqrt(L1*L1 - P1*P1);
float h2 = sqrt(L3*L3 - P2*P2);
float h3 = sqrt(L5*L5 - P3*P3);

float thetaL1 = asin(h1/L1) - ( acos((Ma*Ma + L1*L1 -
Ba*Ba)/(2*Ma*L1)));

float thetaL2 = asin(h1/L2) - ( acos((Ma*Ma + L2*L2 -

```

```

Ba*Ba)/(2*Ma*L2)));

float thetaL3 = asin(h2/L3) - ( acos((Ma*Ma + L3*L3 -
Ba*Ba)/(2*Ma*L3)));

float thetaL4 = asin(h2/L4) - ( acos((Ma*Ma + L4*L4 -
Ba*Ba)/(2*Ma*L4)));

float thetaL5 = asin(h3/L5) - ( acos((Ma*Ma + L5*L5 -
Ba*Ba)/(2*Ma*L5)));

float thetaL6 = asin(h3/L6) - ( acos((Ma*Ma + L6*L6 -
Ba*Ba)/(2*Ma*L6)));

printf("thetaL1:%lf thetaL2:%lf thetaL3:%lf thetaL4:%lf thetaL5:%lf thetaL6:%lf
\n",
thetaL1*rad2deg,thetaL2*rad2deg,thetaL3*rad2deg,thetaL4*rad2deg,thetaL5*rad2de
g,thetaL6*rad2deg);

//Conversão de ângulo para valor de motor

thetaL1 = (-(thetaL1*rad2deg/pos_resolution)+ref);
thetaL2 = ((thetaL2*rad2deg/pos_resolution)+ref);
thetaL3 = (-(thetaL3*rad2deg/pos_resolution)+ref);
thetaL4 = ((thetaL4*rad2deg/pos_resolution)+ref);
thetaL5 = (-(thetaL5*rad2deg/pos_resolution)+ref);
thetaL6 = ((thetaL6*rad2deg/pos_resolution)+ref);

data[0] = (uint16_t)thetaL1;
com_port.writeData((uint8_t) ID1, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

data[0] = (uint16_t)thetaL2;
com_port.writeData((uint8_t) ID2, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

data[0] = (uint16_t)thetaL3;
com_port.writeData((uint8_t) ID3, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

```

```

        data[0] = (uint16_t)thetaL4;
        com_port.writeData((uint8_t) ID4, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

        data[0] = (uint16_t)thetaL5;
        com_port.writeData((uint8_t) ID5, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

        data[0] = (uint16_t)thetaL6;
        com_port.writeData((uint8_t) ID0, P_GOAL_POSITION_L, 4, (uint8_t*)
&data[0]);

        usleep(100000);
        t = t+1;
        if(t>300) t = 0;
    }

    return(0);
}

```

```

void lk(double *L, double px, double py, double pz, double alpha, double beta, double
gamma){

```

```

    float Xt1 = px + a*(sin(alpha)*sin(beta)*sin(gamma + pi/3) +
cos(beta)*cos(gamma + pi/3))/sqrt(3);

```

```

    float Xt2 = px - a*(sin(alpha)*sin(beta)*sin(gamma) +
cos(beta)*cos(gamma))/sqrt(3);

```

```

    float Xt3 = px + a*(sin(alpha)*sin(beta)*sin(gamma - pi/3) +
cos(beta)*cos(gamma - pi/3))/sqrt(3);

```

```

    float Yt1 = py + a*(cos(alpha)*sin(gamma + pi/3))/sqrt(3);

```

```

float Yt2 = py - a*(cos(alpha)*sin(gamma))/sqrt(3);

float Yt3 = py + a*(cos(alpha)*sin(gamma - pi/3))/sqrt(3);

float Zt1 = pz + a*(sin(alpha)*cos(beta)*sin(gamma + pi/3) -
sin(beta)*cos(gamma + pi/3))/sqrt(3);

float Zt2 = pz - a*(sin(alpha)*cos(beta)*sin(gamma) -
sin(beta)*cos(gamma))/sqrt(3);

float Zt3 = pz + a*(sin(alpha)*cos(beta)*sin(gamma - pi/3) -
sin(beta)*cos(gamma - pi/3))/sqrt(3);

*(L+0) = sqrt((Xt1 - d/(2*sqrt(3)) - b/sqrt(3))*(Xt1 - d/(2*sqrt(3)) - b/sqrt(3)) +
(Yt1 - d/2)*(Yt1 - d/2) + Zt1*Zt1);

*(L+1) = sqrt((Xt1 - d/(2*sqrt(3)) + b/(2*sqrt(3)))*(Xt1 - d/(2*sqrt(3)) +
b/(2*sqrt(3))) + (Yt1 - d/2 - b/2)*(Yt1 - d/2 - b/2) + Zt1*Zt1);

*(L+2) = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))*(Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))
+ (Yt2 - b/2)*(Yt2 - b/2) + Zt2*Zt2);

*(L+3) = sqrt((Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))*(Xt2 + d/(sqrt(3)) + b/(2*sqrt(3)))
+ (Yt2 + b/2)*(Yt2 + b/2) + Zt2*Zt2);

*(L+4) = sqrt((Xt3 - d/(2*sqrt(3)) + b/(2*sqrt(3)))*(Xt3 - d/(2*sqrt(3)) +
b/(2*sqrt(3))) + (Yt3 + b/2 + d/2)*(Yt3 + b/2 + d/2) + Zt3*Zt3);

*(L+5) = sqrt((Xt3 - d/(2*sqrt(3)) - b/sqrt(3))*(Xt3 - d/(2*sqrt(3)) - b/sqrt(3)) +
(Yt3 + d/2)*(Yt3 + d/2) + Zt3*Zt3);

}

void Fk(double* px, double* py, double* pz, double* alpha, double* beta, double*
gamma, double *L){

    int ntrial = 1000;

    float errtol_F = 1e-2;

    float errtol_X = 1e-3;

    float err_X = 1;

    float err_F = 1;

```

```
int n = 0;
```

```
double L1 = *(L+0);
```



```

double L2 = *(L+1);
double L3 = *(L+2);
double L4 = *(L+3);
double L5 = *(L+4);
double L6 = *(L+5);

//~ printf("Legs: L1:%lf L2:%lf L3:%lf L4:%lf L5:%lf L6:%lf \n\n",
L1,L2,L3,L4,L5,L6);

float P1 = (b*b + L1*L1 - L2*L2)/(2*b);
float P2 = (b*b + L3*L3 - L4*L4)/(2*b);
float P3 = (b*b + L5*L5 - L6*L6)/(2*b);

float h1 = sqrt(L1*L1 - P1*P1);
float h2 = sqrt(L3*L3 - P2*P2);
float h3 = sqrt(L5*L5 - P3*P3);

float Xp1 = sqrt(3)/6*(2*b + d - 3*P1);
float Xp2 = - sqrt(3)/6*(b + 2*d);
float Xp3 = - sqrt(3)/6*(b-d-3*P3);

float Yp1 = (d+P1)/2;
float Yp2 = (b-2*P2)/2;
float Yp3 = -(b+d-P3)/2;

float Xt1 = Xp1;

```

```

float Xt2 = Xp2;

float Xt3 = Xp3;

//~ printf("%f %f %f \n",Yp1,Yp2,Yp3);

//~ printf("%f %f %f \n",Xp1,Xp2,Xp3);

while((err_F > errtol_F || err_X > errtol_X) && n < ntrial){

    arma::mat F(3,1);

    F(0,0) = a*a + 2*Xt1*Xt2 - 2*Xt1*(Xp1 + sqrt(3)*(Yp1 - Yp2)) -
    2*Xp2*Xt2 - ((sqrt(3)*Xp1 - Yp1 + Yp2)*(sqrt(3)*Xp1 - Yp1 + Yp2) + (h1*h1 + h2*h2) -
    4*Xp1*Xp1 - Xp2*Xp2) + 2*sqrt((h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1))*(h2*h2 - (Xt2 -
    Xp2)*(Xt2 - Xp2)));

    F(1,0) = a*a - 4*Xt1*Xt3 - 2*Xt1*(Xp1 - 3*Xp3 + sqrt(3)*(Yp1 - Yp3))-
    2*Xt3*(-3*Xp1 + Xp3 + sqrt(3)*(Yp1-Yp3)) - ((sqrt(3)*(Xp1 + Xp3) - Yp1 +
    Yp3)*(sqrt(3)*(Xp1 + Xp3) - Yp1 + Yp3) + (h1*h1 + h3*h3) - 4*Xp1*Xp1 - 4*Xp3*Xp3)
    + 2*sqrt((h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1))*(h3*h3 - 4*(Xt3 - Xp3)*(Xt3 - Xp3)));

    F(2,0) = a*a + 2*Xt2*Xt3 - 2*Xt3*(Xp3 + sqrt(3)*(Yp2 - Yp3)) -
    2*Xp2*Xt2 - ((sqrt(3)*Xp3 - Yp2 + Yp3)*(sqrt(3)*Xp3 - Yp2 + Yp3) + (h2*h2 + h3*h3) -
    4*Xp3*Xp3 - Xp2*Xp2) + 2*sqrt((h3*h3 - 4*(Xt3-Xp3)*(Xt3-Xp3))*(h2*h2 - (Xt2 -
    Xp2)*(Xt2 - Xp2)));

    //~ printf("F1:%lf F2:%lf F3:%lf\n\n", F(0,0), F(1,0), F(2,0));

    arma::mat J(3,3);

    //J11 = diff(F1,Xt1);

    J(0, 0) = 2*Xt2 - 2*Xp1 - 2*sqrt(3)*(Yp1 - Yp2) + ((h2*h2 - (Xp2 -
    Xt2)*(Xp2 - Xt2))*(8*Xp1 - 8*Xt1))/sqrt(((h1*h1 - 4*(Xp1 - Xt1)*(Xp1 - Xt1))*(h2*h2 -
    (Xp2 - Xt2)*(Xp2 - Xt2))));

    //J12 = diff(F1,Xt2);

    J(0, 1) = 2*Xt1 - 2*Xp2 + ((h1*h1 - 4*(Xp1 - Xt1)*(Xp1 - Xt1))*(2*Xp2 -
    2*Xt2))/sqrt((h1*h1 - 4*(Xp1 - Xt1)*(Xp1 - Xt1))*(h2*h2 - (Xp2 - Xt2)*(Xp2 - Xt2)));

    //J13 = diff(F1,Xt3);

    J(0, 2) = 0;

    //J21 = diff(F2,Xt1);

    J(1, 0) = 6*Xp3 - 2*Xp1 - 4*Xt3 - 2*sqrt(3)*(Yp1 - Yp3) + ((h3*h3 -

```

```

4*(Xp3 - Xt3)*(Xp3 - Xt3))*(8*Xp1 - 8*Xt1))/sqrt(((h1*h1 - 4*(Xp1 - Xt1)*(Xp1 -
Xt1))*(h3*h3 - 4*(Xp3 - Xt3)*(Xp3 - Xt3))));

//J22 = diff(F2,Xt2);

J(1, 1) = 0;

//J23 = diff(F2,Xt3);

J(1, 2) = 6*Xp1 - 2*Xp3 - 4*Xt1 - 2*sqrt(3)*(Yp1 - Yp3) + ((h1*h1 -
4*(Xp1 - Xt1)*(Xp1 - Xt1))*(8*Xp3 - 8*Xt3))/sqrt(((h1*h1 - 4*(Xp1 - Xt1)*(Xp1 -
Xt1))*(h3*h3 - 4*(Xp3 - Xt3)*(Xp3 - Xt3))));

//J31 = diff(F3,Xt1);

J(2, 0) = 0;

//J32 = diff(F3,Xt2);

J(2, 1) = 2*Xt3 - 2*Xp2 + ((h3*h3 - 4*(Xp3 - Xt3)*(Xp3 - Xt3))*(2*Xp2 -
2*Xt2))/sqrt((h2*h2 - (Xp2 - Xt2)*(Xp2 - Xt2))*(h3*h3 - 4*(Xp3 - Xt3)*(Xp3 - Xt3)));

//J33 = diff(F3,Xt3);

J(2, 2) = 2*Xt2 - 2*Xp3 - 2*sqrt(3)*(Yp2 - Yp3) + ((h2*h2 - (Xp2 -
Xt2)*(Xp2 - Xt2))*(8*Xp3 - 8*Xt3))/sqrt((h2*h2 - (Xp2 - Xt2)*(Xp2 - Xt2))*(h3*h3 -
4*(Xp3 - Xt3)*(Xp3 - Xt3)));

//~ printf("%lf %lf %lf
\n%lf %lf %lf\n%lf %lf %lf\n\n",J(0,0),J(0,1),J(0,2),J(1,0),J(1,1),J(1,2),J(2,0),J(2,1),J(2,
2));

arma::mat deltaX(1,3);

F = -F;

J = arma::inv(J);

deltaX(0,0) = J(0,0)*F(0,0) + J(0,1)*F(1,0) + J(0,2)*F(2,0) ;
deltaX(0,1) = J(1,0)*F(0,0) + J(1,1)*F(1,0) + J(1,2)*F(2,0) ;
deltaX(0,2) = J(2,0)*F(0,0) + J(2,1)*F(1,0) + J(2,2)*F(2,0) ;

Xt1 = Xt1 + deltaX(0,0);

```

```

Xt2 = Xt2 + deltaX(0,1);
Xt3 = Xt3 + deltaX(0,2);
//~ printf("%f %f %f \n\n",Xt1,Xt2,Xt3);

for(int i=0;i<3;i++){
    if(deltaX(0,i)<0) deltaX(0,i) = -deltaX(0,i);
    if(F(i,0)<0) F(i,0) = -F(i,0);
}
err_X = deltaX(0,0) + deltaX(0,1) + deltaX(0,2);
err_F = F(0,0) + F(1,0) + F(2,0);
//~ printf("%f %f \n\n",err_X,err_F);
n = n + 1;
//~ printf("%f %f %d\n",err_F, err_X,n);
}

float Yt1 = sqrt(3)*Xt1 - (sqrt(3)*Xp1 - Yp1);
float Yt2 = Yp2;
float Yt3 = - sqrt(3)*Xt3 + (sqrt(3)*Xp3 + Yp3);

float Zt1 = sqrt(h1*h1 - 4*(Xt1-Xp1)*(Xt1-Xp1));
float Zt2 = sqrt(h2*h2 - (Xt2-Xp2)*(Xt2-Xp2));
float Zt3 = sqrt(h3*h3 - 4*(Xt3-Xp3)*(Xt3-Xp3));

*(px) = (Xt1 + Xt2 + Xt3)/3;
*(py) = (Yt1 + Yt2 + Yt3)/3;
*(pz) = (Zt1 + Zt2 + Zt3)/3;

*(gamma) = (double) atan2((Yt1 - 2*Yt2 + Yt3),(sqrt(3)*(Yt1 - Yt3)));

```

```

        *(alpha) = (double) atan2((-2*(Xt1*(Zt3 - Zt2) + Xt2*(Zt1 - Zt3) - Xt3*(Zt1
- Zt2))), (a*(Yt1 - 2*Yt2 + Yt3)*sin(*(gamma)) + sqrt(3)*a*(Yt1 - Yt3)*cos(*(gamma))));

        *(beta) = (double) atan2((- (Zt1 - 2*Zt2 + Zt3)*cos(*(gamma)) +
sqrt(3)*(Zt1-Zt3)*sin(*(gamma))), ((Xt1 - 2*Xt2 + Xt3)*cos(*(gamma)) - sqrt(3)*(Xt1 -
Xt3)*sin(*(gamma))));
    }

```

Channel.cpp

```
#include <stdint.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include "Channel.hpp"
```

```
#include "USB2Dxl.h"
```

```
//INICIALIZAÇÃO
```

```
int Channel::init(uint8_t controller, uint8_t baud_rate, uint8_t port)
```

```
{
```

```
    switch(controller)
```

```
    {
```

```
        case USB2DXL:
```

```
            u2d::inicializar_motores();
```

```
            readData_ = &(u2d::le_dado);
```

```
            writeData_ = &(u2d::escreve_dado);
```

```
            break;
```

```
        default: printf("Default in Channel::init(uint8_t, uint8_t, uint8_t)\n");
```

```
    }
```

```

        return 0;
    }

//Definido para não aparecer erros na finalização de um programa por exemplo
//Se preocupa em finalizar a thread de controle de acesso adequadamente
Channel::~~Channel()
{
    printf("Channel Destruction\n");
}

Channel::Channel()
{
    printf("Construtor padrão Channel\n");
}

Channel::Channel(uint8_t control, uint8_t baud_rate, uint8_t port)//port=255, testa
vários números de portas
{
    Channel::init(control, baud_rate, port);
}

int Channel::readData(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data)
{
    (*readData_)(id, address_init, size, data);
    return 0;
}

int Channel::writeData(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data)
{

```

```

        (*writeData_)(id, address_init, size, data);
    return 0;
}

```

Channel.hpp

```

#ifndef CHANNEL_H
#define CHANNEL_H

#include <cstdint>

#include <semaphore.h>
#include <queue>
#include <mutex>
#include <vector>
#include <thread>

//Constantes que definem o controlador
#define USB2DXL 0

class Channel
{
    private:
        //Ponteiros das funções de comunicação
        //Deve apontar para uma função de acordo com o controlador utilizado
        //(as definições de cada termo são explicados nos métodos públicos correspondentes)
        int (*readData_)(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data);
        int (*writeData_)(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data);

```

```

public:

//ROTINAS DE INICIALIZAÇÃO

Channel(); //Tenta inicializar pelo USB2Dxl

/*

* control: controlador (constante definidas no início deste cabeçalho)

* baud_rate: baud_rate de comunicação dos motores ligados à porta (pode
não ser aplicável)

* port: número da porta do dispositivo (ttyUSB0, ACM1). port=255, testa
vários números de portas

*/

//TODO: port e baud_rate ainda não implementados

Channel(uint8_t controller, uint8_t baud_rate = 1, uint8_t port =
255); //port=255, testa vários números de portas

~Channel();

int init(uint8_t controller, uint8_t baud_rate = 1, uint8_t port = 255); //vide
construtor


//MÉTODOS PRINCIPAIS

//Parâmetros comuns

/*

* id: id do motor a se comunicar (não aplicável a sync_write)

* address_init: endereço do motor inicial a partir do qual se deseja
escrever/ler (endereços em Constants.h)

* size: número de bytes a ser lido/escrito a partir do endereço inicial (em
sync_write )

* *data: onde armazenar/[está armazenado] os dados lidos/[a serem escritos].
No caso da leitura, deve ter espaço de no mínimo "size"

*/

```

```

//Uma função para cada tipo de instrução

//http://support.robotis.com/en/product/dynamixel/communication/dxl_instruction.htm

int readData(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data);

int writeData(uint8_t id, uint8_t address_init, uint8_t size, uint8_t* data);

};

#endif

```

Funcmoveu2d.cpp

```

#include "dynamixel.h"

#include <stdio.h>

#include <stdlib.h>

extern int giBusUsing;

namespace u2d
{
    int escreve_dado (unsigned char id, unsigned char endereco_inicial, unsigned char comprimento, unsigned char* palavras)
    {
        int i;

        dxl_set_txpacket_id(id);

        dxl_set_txpacket_instruction(INST_WRITE);

        dxl_set_txpacket_length(comprimento+3);

        dxl_set_txpacket_parameter(0, endereco_inicial);
    }
}

```

```

        for(i=0; i<comprimento; i++) dxl_set_txpacket_parameter(i+1,
palavras[i]);

        dxl_tx_packet();

        giBusUsing = 0;

        if(dxl_get_result()!=COMM_TXSUCCESS) return 128+dxl_get_result();

        return 0;

    }

    /* Função para ler dados do motor */

    int le_dado (unsigned char id, unsigned char endereco_inicial, unsigned char
comprimento, unsigned char* resposta)

    {

        int i;

        dxl_set_txpacket_id(id);

        dxl_set_txpacket_instruction(INST_READ);

        dxl_set_txpacket_length(4);

        dxl_set_txpacket_parameter(0, endereco_inicial);

        dxl_set_txpacket_parameter(1, comprimento);

        dxl_ttrx_packet();

        for(i=0; i<comprimento; i++)

            resposta[i] = (unsigned char)dxl_get_rxpacket_parameter(i);

        if (dxl_get_result()!=COMM_RXSUCCESS) return 128+dxl_get_result();

        return get_error();

    }

    void inicializar_motores()

    {

        if(dxl_initialize(0, 1)==0)

            exit(-1);

```

```

    }

}

USB2Dxl.hpp

#ifndef USB2DXL_H
#define USB2DXL_H

namespace u2d
{
    /* le_dado: função que copia dados do motor para uma região de memória do
    controlador

    * Parâmetros:

    * id = id do motor desejado (nunca broadcast)

    * endereco_inicial = endereco do primeiro byte lido nos registradores do
motor

    * comprimento = tamanho, em bytes, do bloco de dados lido

    * resposta = endereco de destino da cópia

    * Retorno: 0 se sem erro, bits de erro da comunicação e do motor

    */

    int le_dado (unsigned char id, unsigned char endereco_inicial, unsigned char
comprimento, unsigned char* resposta);

    /* escreve_dado: Função que copia dados da memória do controlador para o
motor

    * Parâmetros:

    * id = id do motor desejado

    * endereco_inicial = endereco do primeiro byte a ser escrito no motor

    * comprimento = tamanho dos dados a serem escritos, em bytes

    * palavras = ponteiro para o primeiro byte dos dados no computador

```

```

    * Retorno: bits de erro do motor, 0 se sem erro

    */

    int escreve_dado (unsigned char id, unsigned char endereco_inicial, unsigned
char comprimento, unsigned char* palavras);

    /* sync_write: Função que envia um pacote para escrever em vários motores
ao mesmo tempo

    * Os dados já devem estar na forma do sync_write (vide datasheet) e o 13
tratado

    * Para tratá-los, pode usar sync_geral

    * Parâmetros:

    * n_motores = número de motores que serão escritos

    * comprimento = comprimento do bloco a ser escrito em cada motor

    * endereco_inicial = endereço do primeiro byte a ser escrito em todos os
motores

    * dados = ponteiro para os dados a serem jogados diretamente no pacote
(vide datasheet)

    * Retorno: erro

    */

    void inicializar_motores();

} //namespace bracket

#endif

```

Dynamixel.c

```

#include "dxl_hal.h"

#include "dynamixel.h"

```

```

#define ID (2)
#define LENGTH (3)
#define INSTRUCTION (4)
#define ERBIT (4)
#define PARAMETER (5)
#define DEFAULT_BAUDNUMBER (1)

```

```

unsigned char gbInstructionPacket[MAXNUM_TXPARAM+10] = {0};
unsigned char gbStatusPacket[MAXNUM_RXPARAM+10] = {0};
unsigned char gbRxPacketLength = 0;
unsigned char gbRxGetLength = 0;
int gbCommStatus = COMM_RXSUCCESS;
int giBusUsing = 0;

```

```

int dxl_initialize(int deviceIndex, int baudnum )
{
    float baudrate;
    baudrate = 2000000.0f / (float)(baudnum + 1);

    if( dxl_hal_open(deviceIndex, baudrate) == 0 )
        return 0;

    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;
    return 1;
}

```

```
}
```

```
void dxl_terminate(void)
```

```
{
```

```
    dxl_hal_close();
```

```
}
```

```
void dxl_tx_packet(void)
```

```
{
```

```
    unsigned char i;
```

```
    unsigned char TxNumByte, RealTxNumByte;
```

```
    unsigned char checksum = 0;
```

```
    if( giBusUsing == 1 )
```

```
        return;
```

```
    giBusUsing = 1;
```

```
    if( gbInstructionPacket[LENGTH] > (MAXNUM_TXPARAM+2) )
```

```
    {
```

```
        gbCommStatus = COMM_TXERROR;
```

```
        giBusUsing = 0;
```

```
        return;
```

```
    }
```

```
    if( gbInstructionPacket[INSTRUCTION] != INST_PING
```

```
        && gbInstructionPacket[INSTRUCTION] != INST_READ
```

```

    && gbInstructionPacket[INSTRUCTION] != INST_WRITE
    && gbInstructionPacket[INSTRUCTION] != INST_REG_WRITE
    && gbInstructionPacket[INSTRUCTION] != INST_ACTION
    && gbInstructionPacket[INSTRUCTION] != INST_RESET
    && gbInstructionPacket[INSTRUCTION] != INST_SYNC_WRITE )
{
    gbCommStatus = COMM_TXERROR;
    giBusUsing = 0;
    return;
}

gbInstructionPacket[0] = 0xff;
gbInstructionPacket[1] = 0xff;
for( i=0; i<(gbInstructionPacket[LENGTH]+1); i++ )
    checksum += gbInstructionPacket[i+2];
gbInstructionPacket[gbInstructionPacket[LENGTH]+3] = ~checksum;

if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus ==
COMM_RXCORRUPT )
    dxl_hal_clear();

TxNumByte = gbInstructionPacket[LENGTH] + 4;

RealTxNumByte = dxl_hal_tx( (unsigned char*)gbInstructionPacket,
TxNumByte );

if( TxNumByte != RealTxNumByte )
{
    gbCommStatus = COMM_TXFAIL;

```

```

        giBusUsing = 0;
        return;
    }

    if( gbInstructionPacket[INSTRUCTION] == INST_READ )
        dxl_hal_set_timeout( gbInstructionPacket[PARAMETER+1] + 6 );
    else
        dxl_hal_set_timeout( 6 );

    gbCommStatus = COMM_TXSUCCESS;
}

void dxl_rx_packet(void)
{
    unsigned char i, j, nRead;
    unsigned char checksum = 0;

    if( giBusUsing == 0 )
        return;

    if( gbInstructionPacket[ID] == BROADCAST_ID )
    {
        gbCommStatus = COMM_RXSUCCESS;
        giBusUsing = 0;
        return;
    }
}

```

```

if( gbCommStatus == COMM_TXSUCCESS )
{
    gbRxGetLength = 0;
    gbRxPacketLength = 6;
}

nRead = dxl_hal_rx( (unsigned char*)&gbStatusPacket[gbRxGetLength],
gbRxPacketLength - gbRxGetLength );

gbRxGetLength += nRead;

if( gbRxGetLength < gbRxPacketLength )
{
    if( dxl_hal_timeout() == 1 )
    {
        if(gbRxGetLength == 0)
            gbCommStatus = COMM_RXTIMEOUT;
        else
            gbCommStatus = COMM_RXCORRUPT;

        giBusUsing = 0;
        return;
    }
}

// Find packet header
for( i=0; i<(gbRxGetLength-1); i++ )
{
    if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
    {

```

```

        break;
    }
    else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] ==
0xff )
    {
        break;
    }
}
if( i > 0 )
{
    for( j=0; j<(gbRxGetLength-i); j++ )
        gbStatusPacket[j] = gbStatusPacket[j + i];

    gbRxGetLength -= i;
}

if( gbRxGetLength < gbRxPacketLength )
{
    gbCommStatus = COMM_RXWAITING;
    return;
}

// Check id pairing
if( gbInstructionPacket[ID] != gbStatusPacket[ID])
{
    gbCommStatus = COMM_RXCORRUPT;
    giBusUsing = 0;
}

```

```

        return;
    }

    gbRxPacketLength = gbStatusPacket[LENGTH] + 4;
    if( gbRxGetLength < gbRxPacketLength )
    {
        nRead = dxl_hal_rx( (unsigned
char*)&gbStatusPacket[gbRxGetLength], gbRxPacketLength - gbRxGetLength );
        gbRxGetLength += nRead;
        if( gbRxGetLength < gbRxPacketLength )
        {
            gbCommStatus = COMM_RXWAITING;
            return;
        }
    }

    // Check checksum
    for( i=0; i<(gbStatusPacket[LENGTH]+1); i++ )
        checksum += gbStatusPacket[i+2];
    checksum = ~checksum;

    if( gbStatusPacket[gbStatusPacket[LENGTH]+3] != checksum )
    {
        gbCommStatus = COMM_RXCORRUPT;
        giBusUsing = 0;
        return;
    }

```

```
    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;
}

void dxl_trx_packet(void)
{
    dxl_tx_packet();

    if( gbCommStatus != COMM_TXSUCCESS )
        return;

    do{
        dxl_rx_packet();
    }while( gbCommStatus == COMM_RXWAITING );
}

int dxl_get_result(void)
{
    return gbCommStatus;
}

void dxl_set_txpacket_id( int id )
{
    gbInstructionPacket[ID] = (unsigned char)id;
}
```

```
void dxl_set_txpacket_instruction( int instruction )
{
    gbInstructionPacket[INSTRUCTION] = (unsigned char)instruction;
}
```

```
void dxl_set_txpacket_parameter( int index, int value )
{
    gbInstructionPacket[PARAMETER+index] = (unsigned char)value;
}
```

```
void dxl_set_txpacket_length( int length )
{
    gbInstructionPacket[LENGTH] = (unsigned char)length;
}
```

```
int dxl_get_rxpacket_error( int errbit )
{
    if( gbStatusPacket[ERRBIT] & (unsigned char)errbit )
        return 1;

    return 0;
}
```

```
int get_error(){return (int)gbStatusPacket[ERRBIT];}
```

```
int dxl_get_rxpacket_length(void)
{
```

```
        return (int)gbStatusPacket[LENGTH];
    }

int dxl_get_rxpacket_parameter( int index )
{
    return (int)gbStatusPacket[PARAMETER+index];
}

int dxl_makeword( int lowbyte, int highbyte )
{
    unsigned short word;

    word = highbyte;
    word = word << 8;
    word = word + lowbyte;
    return (int)word;
}

int dxl_get_lowbyte( int word )
{
    unsigned short temp;

    temp = word & 0xff;
    return (int)temp;
}

int dxl_get_highbyte( int word )
```

```

{
    unsigned short temp;

    temp = word & 0xff00;
    temp = temp >> 8;
    return (int)temp;
}

```

```

void dxl_ping( int id )
{
    while(giBusUsing);

    gbInstructionPacket[ID] = (unsigned char)id;
    gbInstructionPacket[INSTRUCTION] = INST_PING;
    gbInstructionPacket[LENGTH] = 2;

    dxl_txrx_packet();
}

```

```

int dxl_read_byte( int id, int address )
{
    while(giBusUsing);

    gbInstructionPacket[ID] = (unsigned char)id;
    gbInstructionPacket[INSTRUCTION] = INST_READ;
    gbInstructionPacket[PARAMETER] = (unsigned char)address;
    gbInstructionPacket[PARAMETER+1] = 1;
}

```

```

    gbInstructionPacket[LENGTH] = 4;

    dxl_txrx_packet();

    return (int)gbStatusPacket[PARAMETER];
}

void dxl_write_byte( int id, int address, int value )
{
    while(giBusUsing);

    gbInstructionPacket[ID] = (unsigned char)id;
    gbInstructionPacket[INSTRUCTION] = INST_WRITE;
    gbInstructionPacket[PARAMETER] = (unsigned char)address;
    gbInstructionPacket[PARAMETER+1] = (unsigned char)value;
    gbInstructionPacket[LENGTH] = 4;

    dxl_txrx_packet();
}

int dxl_read_word( int id, int address )
{
    while(giBusUsing);

    gbInstructionPacket[ID] = (unsigned char)id;
    gbInstructionPacket[INSTRUCTION] = INST_READ;
    gbInstructionPacket[PARAMETER] = (unsigned char)address;

```

```

    gbInstructionPacket[PARAMETER+1] = 2;
    gbInstructionPacket[LENGTH] = 4;

    dxl_txrx_packet();

    return dxl_makeword((int)gbStatusPacket[PARAMETER],
        (int)gbStatusPacket[PARAMETER+1]);
}

void dxl_write_word( int id, int address, int value )
{
    while(giBusUsing);

    gbInstructionPacket[ID] = (unsigned char)id;
    gbInstructionPacket[INSTRUCTION] = INST_WRITE;
    gbInstructionPacket[PARAMETER] = (unsigned char)address;
    gbInstructionPacket[PARAMETER+1] = (unsigned
char)dxl_get_lowbyte(value);
    gbInstructionPacket[PARAMETER+2] = (unsigned
char)dxl_get_highbyte(value);
    gbInstructionPacket[LENGTH] = 5;

    dxl_txrx_packet();
}

```

Dynamixel.h

```

#ifndef _DYNAMIXEL_HEADER
#define _DYNAMIXEL_HEADER

```

```

#ifdef __cplusplus
extern "C" {
#endif

////////// device control methods //////////

int dxl_initialize(int deviceIndex, int baudnum );
void dxl_terminate();

////////// set/get packet methods //////////

#define MAXNUM_TXPARAM          (150)
#define MAXNUM_RXPARAM          (60)

void dxl_set_txpacket_id(int id);
#define BROADCAST_ID            (254)

void dxl_set_txpacket_instruction(int instruction);
#define INST_PING                (1)
#define INST_READ                (2)
#define INST_WRITE               (3)
#define INST_REG_WRITE           (4)
#define INST_ACTION              (5)
#define INST_RESET               (6)
#define INST_SYNC_WRITE          (131)

void dxl_set_txpacket_parameter(int index, int value);
void dxl_set_txpacket_length(int length);

```

```

int dxl_get_rxpacket_error(int errbit);

#define ERRBIT_VOLTAGE          (1)
#define ERRBIT_ANGLE           (2)
#define ERRBIT_OVERHEAT        (4)
#define ERRBIT_RANGE           (8)
#define ERRBIT_CHECKSUM        (16)
#define ERRBIT_OVERLOAD        (32)
#define ERRBIT_INSTRUCTION     (64)


int dxl_get_rxpacket_length(void);
int dxl_get_rxpacket_parameter(int index);


// utility for value
int dxl_makeword(int lowbyte, int highbyte);
int dxl_get_lowbyte(int word);
int dxl_get_highbyte(int word);


////////// packet communication methods //////////

void dxl_tx_packet(void);
void dxl_rx_packet(void);
void dxl_txrx_packet(void);


int dxl_get_result(void);

#define COMM_TXSUCCESS          (0)
#define COMM_RXSUCCESS          (1)
#define COMM_TXFAIL             (2)

```

```

#define COMM_RXFAIL                (3)
#define COMM_TXERROR                (4)
#define COMM_RXWAITING              (5)
#define COMM_RXTIMEOUT              (6)
#define COMM_RXCORRUPT              (7)

```

```

////////// high communication methods //////////

```

```

void dxl_ping(int id);
int dxl_read_byte(int id, int address);
void dxl_write_byte(int id, int address, int value);
int dxl_read_word(int id, int address);
void dxl_write_word(int id, int address, int value);

```

```

int get_error();

```

```

#ifdef __cplusplus
}
#endif
#endif

```

Dxl_hal.c

```

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

```

```

#include <termios.h>

#include <linux/serial.h>

#include <sys/ioctl.h>

#include <sys/time.h>


#include "dxl_hal.h"


int    gSocket_fd  = -1;
long   glStartTime = 0;
float   gfRcvWaitTime    = 0.0f;
float   gfByteTransTime  = 0.0f;


char   gDeviceName[20];


int dxl_hal_open(int deviceIndex, float baudrate)
{
    struct termios newtio;
    struct serial_struct serinfo;
    char dev_name[100] = {0, };

    int fd;
    for(deviceIndex=0; deviceIndex<10; deviceIndex++) {
        sprintf(dev_name, "/dev/ttyUSB%d", deviceIndex);
        fd = open(dev_name, O_RDONLY);
        if(fd>=0) break;
        close(fd);
    }
}

```

```

close(fd);

sprintf(dev_name, "/dev/ttyUSB%d", deviceIndex);

strcpy(gDeviceName, dev_name);

memset(&newtio, 0, sizeof(newtio));

dxl_hal_close();

if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK))
< 0) {

    fprintf(stderr, "device open error: %s\n", dev_name);

    goto DXL_HAL_OPEN_ERROR;

}

newtio.c_cflag      = B38400|CS8|CLOCAL|CREAD;
newtio.c_iflag      = IGNPAR;
newtio.c_oflag      = 0;
newtio.c_lflag      = 0;
newtio.c_cc[VTIME]= 0;  // time-out   (TIME * 0.1 ) 0 : disable
newtio.c_cc[VMIN] = 0;  // MIN    read    return

tcflush(gSocket_fd, TCIFLUSH);
tcsetattr(gSocket_fd, TCSANOW, &newtio);

if(gSocket_fd == -1)

    return 0;

if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {

```

```

        fprintf(stderr, "Cannot get serial info\n");
        return 0;
    }

    serinfo.flags &= ~ASYNC_SPD_MASK;
    serinfo.flags |= ASYNC_SPD_CUST;
    serinfo.custom_divisor = serinfo.baud_base / baudrate;

    if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
        fprintf(stderr, "Cannot set serial info\n");
        return 0;
    }

    dxl_hal_close();

    gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);

    strcpy(gDeviceName, dev_name);
    memset(&newtio, 0, sizeof(newtio));
    dxl_hal_close();

    if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK))
    < 0) {
        fprintf(stderr, "device open error: %s\n", dev_name);
        goto DXL_HAL_OPEN_ERROR;
    }

```

```

newtio.c_cflag          = B38400|CS8|CLOCAL|CREAD;
newtio.c_iflag           = IGNPAR;
newtio.c_oflag           = 0;
newtio.c_lflag           = 0;
newtio.c_cc[VTIME]= 0;   // time-out   (TIME * 0.1 ) 0 : disable
newtio.c_cc[VMIN] = 0;   // MIN   read   return

```

```

tcflush(gSocket_fd, TCIFLUSH);
tcsetattr(gSocket_fd, TCSANOW, &newtio);

```

```

return 1;

```

DXL_HAL_OPEN_ERROR:

```

    dxl_hal_close();
    return 0;

```

```

}

```

void dxl_hal_close()

```

{
    if(gSocket_fd != -1)
        close(gSocket_fd);
    gSocket_fd = -1;
}

```

int dxl_hal_set_baud(float baudrate)

```

{
    struct serial_struct serinfo;

```

```

    if(gSocket_fd == -1)
        return 0;

    if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {
        fprintf(stderr, "Cannot get serial info\n");
        return 0;
    }

    serinfo.flags &= ~ASYNC_SPD_MASK;
    serinfo.flags |= ASYNC_SPD_CUST;
    serinfo.custom_divisor = serinfo.baud_base / baudrate;

    if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
        fprintf(stderr, "Cannot set serial info\n");
        return 0;
    }

    //dxl_hal_close();

    //dxl_hal_open(gDeviceName, baudrate);

    gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);
    return 1;
}

void dxl_hal_clear(void)
{
    tcflush(gSocket_fd, TCIFLUSH);
}

```

```

int dxl_hal_tx( unsigned char *pPacket, int numPacket )
{
    return write(gSocket_fd, pPacket, numPacket);
}

```

```

int dxl_hal_rx( unsigned char *pPacket, int numPacket )
{
    memset(pPacket, 0, numPacket);
    return read(gSocket_fd, pPacket, numPacket);
}

```

```

static inline long myclock()
{
    struct timeval tv;
    gettimeofday (&tv, NULL);
    return (tv.tv_sec * 1000 + tv.tv_usec / 1000);
}

```

```

void dxl_hal_set_timeout( int NumRcvByte )
{
    glStartTime = myclock();
    gfRcvWaitTime = (float)(gfByteTransTime*(float)NumRcvByte + 5.0f);
}

```

```

int dxl_hal_timeout(void)
{

```

```

long time;

time = myclock() - glStartTime;

if(time > 10)
{
    printf("timeout: %ld %f\n", time, gfRcvWaitTime);
    return 1;
}
else if(time < 0)
{
    printf("rx_ok: %ld %f\n", time, gfRcvWaitTime);
    glStartTime = myclock();
}
//printf("rx_ok: %ld %f\n", time, gfRcvWaitTime);
return 0;
}

```

Dxl_hal.h

```

#ifndef _DYNAMIXEL_HAL_HEADER
#define _DYNAMIXEL_HAL_HEADER

#ifdef __cplusplus
extern "C" {
#endif

int dxl_hal_open(int deviceIndex, float baudrate);

```

```
void dxl_hal_close();  
int dxl_hal_set_baud( float baudrate );  
void dxl_hal_clear();  
int dxl_hal_tx( unsigned char *pPacket, int numPacket );  
int dxl_hal_rx( unsigned char *pPacket, int numPacket );  
void dxl_hal_set_timeout( int NumRcvByte );  
int dxl_hal_timeout();  
#ifdef __cplusplus  
}  
#endif  
#endif
```

ANEXO

ANEXO A – Conjunto de instruções para motores AX-12A.

Kind of Instruction

To operate Dynamixel, Instruction Packet, which is binary type data, should be sent to Dynamixel from Main Controller.

Instruction Packet has seven kinds of commands. (Refer to “Instruction Packet”)

In addition, Dynamixel receives Instruction Packet to perform a command and returns the result as Status Packet to Main Controller. This section describes examples of the usage of each command of Instruction Packet.

The following example is written based on Dynamixel Actuator RX-64. Since other Dynamixels such as AX12/12+, DX etc. consist of equal command, the same Packet type can be used.

READ DATA

Function This command is to read data in the Control Table inside of RX-64.

Length 0X04

Instruction 0X02

Parameter1 Start Address of data to be read

Parameter2 Length of Data to be read

Example 1	Reads the current internal temperature of RX-64 whose ID is 1.
Reads 1 byte from the value of Address 43 (0x2B) in the Control Table.	
Instruction Packet : 0xFF 0xFF 0x01 0x04 0x02 0x2B 0x01 0xCC	
CHECKSUM	
Status Packet returned is as follows:	
Status Packet : 0xFF 0xFF 0x01 0x03 0x00 0x20 0xDB	
Data value read is 0x20 (i.e., 32 in decimal). Thus, the current internal temperature	

WRITE DATA

Function This command is to write data to the Control Table inside of RX-64.

Length N+3 (if the number of writing data is N)

Instruction 0X03

Parameter1 Start address to write data

Parameter2 First data to write

Parameter3 Second data to write

Parameter N+1 Nth Data to write

Example 2 Sets the ID of RX-64 as "1".

Writes 1 to the Address 3 in the Control Table.

Sends ID as Broadcasting ID(0xFE).

Instruction Packet : 0XFF 0XFF 0XFE 0X04 0X03 0X03 0X01 0XF6'

Status Packet is not returned since Broadcast ID (0xFE) is transmitted.

REG WRITE

Function The REG_WRITE command is similar to the WRITE_DATA command in terms of function, but differs in terms of the timing that a command is executed. When Instruction Packet arrives, it is saved in Buffer and the Write operation remains in the standby state. At this moment, Registered Instruction (Address 44 (0x2C)) is set as "1". Then, when Action Instruction Packet arrives, Registered Instruction changes into "0" and the registered Write command is finally executed.

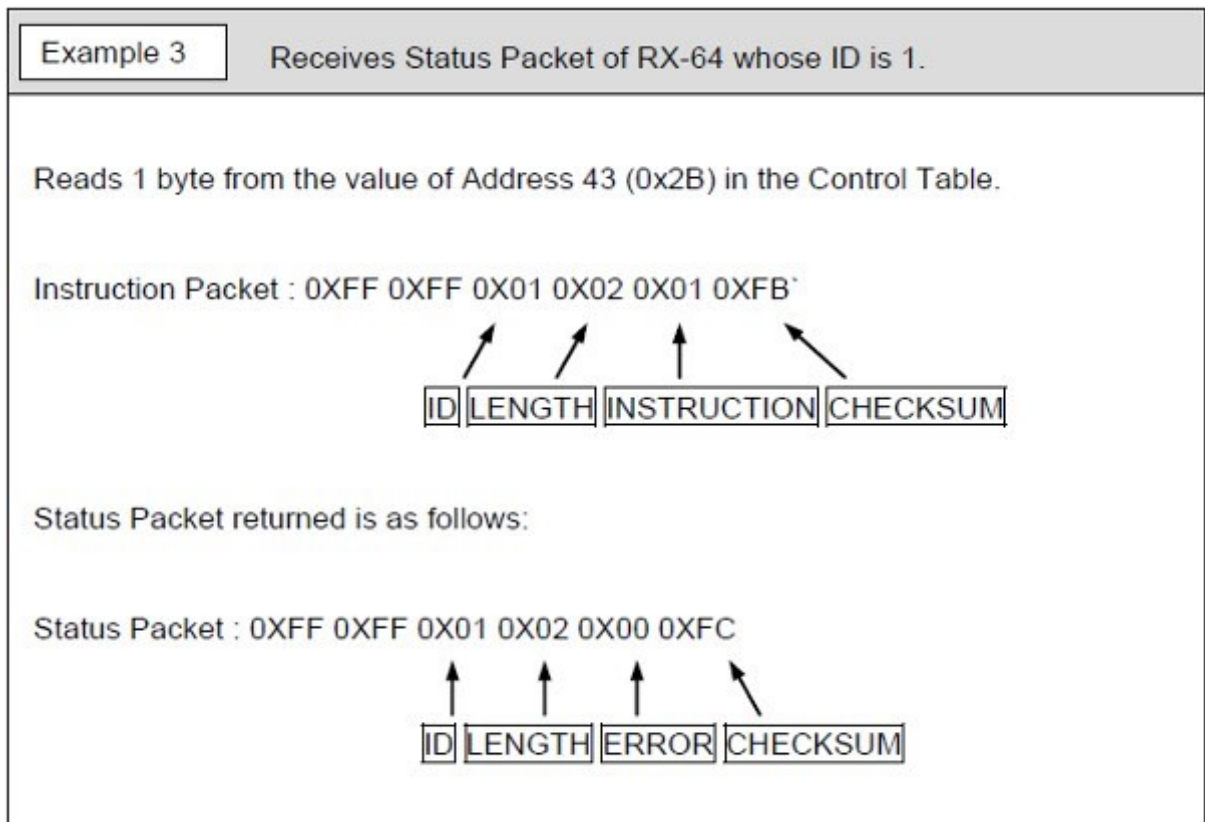
Length N+3 (if the number of Writing Data is N)

Instruction 0X04

Parameter1 Start Address to write Data

Parameter2 First data to write

Parameter N+1 Nth data to write



ACTION

Function This command is to execute the Write action registered by REG_WRITE

Length 0x02

Instruction 0x05

Parameter NONE

The Action command is useful when several RX-64s are moved with accuracy at the same time. When several running gears are controlled via communication, there is a little time difference in terms of enabling time between the first and the last running gear getting commands. RX-64 has resolved this problem by using Action Instruction.

PING

Function This command does not instruct anything. It is only used when receiving Status Packet or confirming the existence of RX-64 with a specific ID.

Length 0x02

Instruction 0x01

Parameter NONE

Although Status Return Level (Address 16 (0X10)) is 0, it returns Status Packet all the time for Ping Instruction. But, it does not return Status Packet when Check Sum Error occurs in spite of using PING Instruction.

RESET

Function This command is to reset the Control Table of RX-64 to the factory default setting

Please be careful since the value set by users can be erased if RESET command is used.

Length 0X02

Instruction 0X06

Parameter NONE

Example 4 Resets the Control Table of RX-64 whose ID is 0.

Instruction Packet : 0XFF 0XFF 0X00 0X02 0X06 0XF7

ID	LENGTH	INSTRUCTION	CHECKSUM
----	--------	-------------	----------

Status Packet returned is as follows:

Status Packet : 0XFF 0XFF 0X00 0X02 0X00 0XFD

ID	LENGTH	ERROR	CHECKSUM
----	--------	-------	----------

Please note that ID is changed into "1" after the execution of the RESET command.

SYNC WRITE

Function This command is used to control several RX-64s simultaneously with one Instruction Packet transmission. When this command is used, several commands are transmitted at once, so that the communication time is reduced when multiple RX-64s are controlled. However, the SYNC WRITE command can be used only if both of the address and length of the Control Table to write is identical. Besides, ID should be transmitted as Broadcasting ID.

Generally, in the event 1 command packet is 4 byte, 26 Dynamixel can be controlled simultaneously. Make sure that the length of packet does not to exceed 143 bytes since the volume of receiving buffer of RX-64 is 143 bytes.

ID 0XFE

Length $(L+1) \times N + 4$ (L: Data Length per RX-64, N: the number of RX-64s)

Instruction 0X83

Parameter1 Start address to write Data

Parameter2 Length of Data to write

Parameter3 First ID of RX-64

Parameter4 First data of the first RX-64

Parameter5 Second data of the first RX-64

...

Parameter L+3 Lth Data of the first RX-64

Parameter L+4 ID of the second RX-64

Parameter L+5 First data of the second RX-64

Parameter L+6 Second data of the second RX-64

...

Parameter 2L+4 Lth data of the second RX-64

Example 5

Moves to the following position and speed for each RX-64.

RX-64 with ID 0 : Moves to the position of 0x010 at the speed of 0x150

RX-64 with ID 1 : Moves to the position of 0x220 at the speed of 0x360

RX-64 with ID 2: Moves to the position of 0x030 at the speed of 0x170

RX-64 with ID 3: Moves to the position of 0x220 at the speed of 0x380

Instruction Packet : 0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00
 0X50 0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00
 0X70 0X01 0X03 0X20 0X02 0X80 0X03 0X12`

Status Packet is not returned since ID is transmitted as Broadcasting ID.

BULK READ (This command only works for MX series)

Function This command is used for reading the values of several DYNAMIXELs simultaneously, by sending a single Instruction Packet. The packet length is lessened compared to sending many READ commands, and the idle time between the status packets being returned is also lessened to save communication time. But, this cannot be used to read many times on a single module, and if several of the same module ID is designated, only the firstly designated parameter will be processed.

ID 0XFE

Length 3N+3

Instruction 0X92

Parameter1 0X00

Parameter2 Length of the data to be read from the first module [L]

Parameter3 ID of the first module

Parameter4 Starting address of the data to be read from the first module

...

Parameter 3N+2 : Length of the data to be read from the Nth module [L]

Parameter 3N+3 : ID of the Nth module

Parameter 3N+4 : Starting address of the data to be read from the Nth module

example)

DYANMIXEL with ID 1 : Brings the goal position value (2 bytes from 0x1E).

DYNAMIXEL with ID 2 : Brings the current position value (2 bytes from 0x24).

The command packet to order this movement is as follows;

0XFF 0XFF 0XFE 0X09 0X92 0X00 0X02 0X01 0X1E 0X02 0X02 0X24 0X1D

During this time, module with ID 2 monitors the status packet being sent from ID 1 of the data bus (the very previous parameter ID), then right when module ID 1's status packet completes transmission, sends its own status packet. The returned status packet is as follows;

0XFF 0XFF 0X01 0X04 0X00 0X00 0X80 0X7A 0XFF 0XFF 0X02 0X04 0X00 0X00 0X80 0X79 Status packets from each of module ID 1 and ID 2 will come in one after another.

Other Example

The following examples are assumed that ID = 1, and Baud rate = 57142 BPS.

Example 6	Reads the Model Number and Firmware Version.
Hint	Instruction = READ_DATA, Address = 0x00, Length = 0x04
Communication	Instruction Packet : FF FF 01 04 02 00 03 F5 Status Packet : FF FF 01 05 00 40 00 08 B1
Status Packet Result	Model Number = 64 (0x40) Firmware Version = 0x08

Example 7

Changes the ID of RX-64 from 1 to 0.

Hint Instruction = WRITE_DATA, Address = 0x03, DATA = 0x00

Communication Instruction Packet : FF FF 01 04 03 03 00 F4
 Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 8

Changes the Baud Rate to 1M bps.

Hint Instruction = WRITE_DATA, Address = 0x04, DATA = 0x01

Communication Instruction Packet : FF FF 01 04 03 04 01 F2
 Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 9

Resets Return Delay Time as 4usec.

Hint Instruction = WRITE_DATA, Address = 0x05,
 DATA = 0x02

Communication Instruction Packet : FF FF 01 04 03 05 02 F0
 Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 10

Restricts the movement angle from 0 to 150°.

Hint

Since CCW Angle Limit 0x3FF means 300°,
150° corresponds to 0x200.

Instruction = WRITE_DATA, Address = 0x08,
DATA = 0x00, 0x02

Communication

Instruction Packet : FF FF 01 05 03 08 00 02 EC
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR**Example 11**

Resets the highest limit of operating temperature as 80°.

Hint

Instruction = WRITE_DATA, Address = 0x0B,
DATA = 0x50

Communication

Instruction Packet : FF FF 01 04 03 0B 50 9C
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR**Example 12**

Sets the operating voltage as 10 to 17V.

Hint

Data of 10V is 100 (0x64) while 17V is 170 (0xAA).
Instruction = WRITE_DATA, Address = 0x0C,
DATA = 0x64, 0xAA

Communication

Instruction Packet : FF FF 01 05 03 0C 64 AA DC
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 13

Only generates 50% of the maximum torque.

Hint
area

Sets the value of MAX Torque located in the EEPROM
as 0x1FF, which is 50% of the maximum value 0x3FF.
Instruction = WRITE_DATA, Address = 0x0E,
DATA = 0xff, 0x01

Communication

Instruction Packet: FF FF 01 05 03 0E FF 01 E8
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

The change of Max Torque can be checked by turning the power off and then on.

Example 15

Sets the Alarm as such that LED flickers and shutdown (torque off)
when the operating temperature is higher than the limit temperature.

Hint

Since Overheating Error is Bit 2, set up Alarm value as
0x04. (0x04=00000100)
Instruction = WRITE_DATA, Address = 0x11,
DATA = 0x04, 0x04

Communication

Instruction Packet: FF FF 01 05 03 11 04 04 DD
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 16

Turns on the LED and enables Torque.

Hint

Instruction = WRITE_DATA, Address = 0x18,
DATA = 0x01, 0x01

Communication

Instruction Packet: FF FF 01 05 03 18 01 01 DC
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

You can check the Torque Enable state by touching the axis of Dynamixel you're your hand.

Example 17

Locates at the Position 180° with the speed of 57RPM.

Hint

Sets Goal Position (Address 30 (0x1E))= 512 (0x200) and
Moving Speed (Address 0x20)= 512 (0x200).
Instruction = WRITE_DATA, Address = 0x1E,
DATA = 0x00, 0x02, 0x00, 0x02

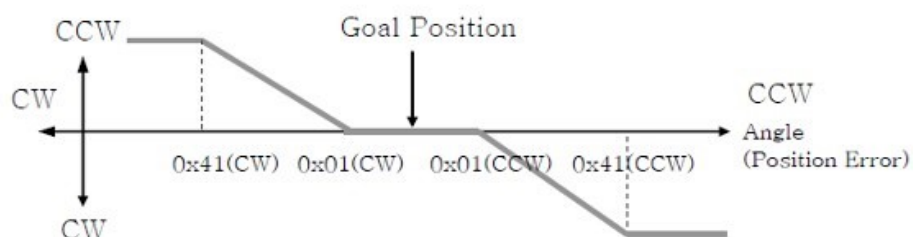
Communication

Instruction Packet: FF FF 01 07 03 1E 00 02 00 02 D2
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 18 Sets Compliance Margin=1 and Compliance Slope=0x40.

Hint The suggested condition can be depicted in a graph as below.



- A: CCW Compliance Slope (Address 29 (0x1D)) = 0x40 (about 18.8°)
 B: CCW Compliance Margin (Address 27 (0x1B)) = 0x01 (about 0.3°)
 C: CW Compliance Margin (Address 26 (0x1A)) = 0x01 (about 0.3°)
 D: CW Compliance Slope (Address 28 (0x1C)) = 0x40 (about 18.8°)

Instruction = WRITE_DATA, Address = 0x1A,
 DATA = 0x01, 0x01, 0x40, 0x40

Communication Instruction Packet: FF FF 01 07 03 1A 01 01 40 40 58
 Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 19 Sets the minimum output Torque (Punch) as 0x40.

Hint Instruction = WRITE_DATA, Address = 0x30,
 DATA = 0x40, 0x00

Communication Instruction Packet : FF FF 01 05 03 30 40 00 86
 Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 20 Locates RX-64 with ID 0 at Position 0° and RX-64 with ID 1 at Position 300°. Start only two RX-64s at the same point.

Hint

When the WRITE_DATA command is used, two RX-64s cannot be started at the same point.

Thus, REG_WRITE and ACTION are used.

ID=0, Instruction = REG_WRITE, Address = 0x1E, DATA = 0x00, 0x00

ID=1, Instruction = REG_WRITE, Address = 0x1E, DATA = 0xff, 0x03

ID=0xfe(Broadcasting ID), Instruction = ACTION,

Communication

Instruction Packet: FF FF 00 05 04 1E 00 00 D8

Status Packet : FF FF 00 02 00 FD

Instruction Packet: FF FF 01 05 04 1E FF 03 D5

Status Packet : FF FF 01 02 00 FC

Instruction Packet: FF FF FE 02 05 FA (LEN:006)

Status Packet //No return packet

Status Packet Result NO ERROR

Example 21 Unable to change values except Address 24 to Address 35.
Hint

Set Lock (Address 47 (0x2F)) as 1.

Instruction = WRITE_DATA, Address = 0x2F,

DATA = 0x01

Communication

Instruction Packet : FF FF 01 04 03 2F 01 C7

Status Packet : FF FF 01 02 00 FC

Status Packet Result

Status Packet Result NO ERROR

Once locked, It is impossible to unlock unless the power is off.

When other data is accessed while locked, an error is returned.

AX Series > AX-12/ AX-12+/ AX-12A ROBOTIS e-Manual v1.29.00

AX-12/ AX-12+/ AX-12A



AX-12+ is the improved version of existing AX-12; the design of circuit, material, and wheel gear are specially improved. ※ AX-12A is a new version of the AX-12+ with the same performance but more advanced external design. Only the AX-12A is now being sold.

H/W Specification

- Weight : 53.5g (AX-12/AX-12+), 54.6g (AX-12A)
- Dimension : 32mm * 50mm * 40mm
- Resolution : 0.29°
- Gear Reduction Ratio : 254 : 1
- Stall Torque : 1.5N.m (at 12.0V, 1.5A)
- No load speed : 59rpm (at 12V)
- Running Degree
 - 0° ~ 300°
 - Endless Turn

Running Temperature : -5°C ~ +70°C

Voltage : 9 ~ 12V (Recommended Voltage 11.1V)

Command Signal : Digital Packet

Protocol Type : Half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)

Link (Physical) : TTL Level Multi Drop (daisy chain type Connector)

ID : 254 ID (0~253)

Communication Speed : 7343bps ~ 1 Mbps

Feedback : Position, Temperature, Load, Input Voltage, etc.

Material : Engineering Plastic

Stall torque is the maximum instantaneous and static torque

Stable motions are possible with robots designed for loads with $1/5$ or less of the stall torque

Control Table

Control Table consists of data regarding the current status and operation, which exists inside of Dynamixel. The user can control Dynamixel by changing data of Control Table via Instruction Packet.

EEPROM and RAM

Data in RAM area is reset to the initial value whenever the power is turned on while data in EEPROM area is kept once the value is set even if the power is turned off.

Address

It represents the location of data. To read from or write data to Control Table, the user should assign the correct address in the Instruction Packet.

Access

Dynamixel has two kinds of data: Read-only data, which is mainly used for sensing, and Read-and-Write data, which is used for driving.

Initial Value

In case of data in the EEPROM Area, the initial values on the right side of the below Control Table are the factory default settings. In case of data in the RAM Area, the initial values on the right side of the above Control Tables are the ones when the power is turned on.

Area	Address (Hexadecimal)	Name	Description
E E P R O M	0 (0X00)	Model Number(L)	Lowest byte of model number
	1 (0X01)	Model Number(H)	Highest byte of model number
	2 (0X02)	Version of Firmware	Information on the version of firmware
	3 (0X03)	ID	ID of Dynamixel
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel
	5 (0X05)	Return Delay Time	Return Delay Time
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque
	16 (0X10)	Status Return Level	Status Return Level
	17 (0X11)	Alarm LED	LED for Alarm
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm
R A	24 (0X18)	Torque Enable	Torque On/Off
	25 (0X19)	LED	LED On/Off
	26 (0X1A)	CW Compliance Margin	CW Compliance margin
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin
	28 (0X1C)	CW Compliance Slope	CW Compliance slope
	29 (0X1D)	CCW Compliance Slope	CCW Compliance slope
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed (Moving Velocity)
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed (Moving Velocity)
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit (Goal Torque)
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit (Goal Torque)

M	36 (0X24)	Present Position(L)	Lowest byte of Current Position (Present Velocity)
	37 (0X25)	Present Position(H)	Highest byte of Current Position (Present Velocity)
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed
	40 (0X28)	Present Load(L)	Lowest byte of Current Load
	41 (0X29)	Present Load(H)	Highest byte of Current Load
	42 (0X2A)	Present Voltage	Current Voltage
	43 (0X2B)	Present Temperature	Current Temperature
	44 (0X2C)	Registered	Means if Instruction is registered
	46 (0X2E)	Moving	Means if there is any movement
	47 (0X2F)	Lock	Locking EEPROM
	48 (0X30)	Punch(L)	Lowest byte of Punch
	49 (0X31)	Punch(H)	Highest byte of Punch

Highest/Lowest Byte

In the Control table, some data share the same name, but they are attached with (L) or (H) at the end of each name to distinguish the address. This data requires 16bit, but it is divided into 8bit each for the addresses (low) and (high). These two addresses should be written with one Instruction Packet at the same time.

Address Function Help

EEPROM Area Model Number

It represents the Model Number.

Firmware Version

It represents the firmware version.

ID

It is a unique number to identify Dynamixel.

The range from 0 to 252 (0xFC) can be used, and, especially, 254(0xFE) is used as

the Broadcast ID.

If the Broadcast ID is used to transmit Instruction Packet, we can command to all Dynamixels.

Please be cautious not to have the same IDs for the connected dynamixels. You may face communication issues or may not be able to search when IDs overlap.

Baud Rate

It represents the communication speed. 0 to 254 (0xFE) can be used for it.

This speed is calculated by using the below formula.

Data	Set BPS	Target BPS	Tolerance
207	9615.4	9600.0	-0.160 %

Note : Maximum Baud Rate error of 3% is within the tolerance of UART communication.

Return Delay Time

It is the delay time per data value that takes from the transmission of Instruction Packet until the return of Status Packet.

0 to 254 (0xFE) can be used, and the delay time per data value is 2 usec.

That is to say, if the data value is 10, 20 usec is delayed. The initial value is 250 (0xFA) (i.e., 0.5 msec).

CW/CCW Angle Limit

The angle limit allows the motion to be restrained.

The range and the unit of the value is the same as Goal Position(Address 30, 31).

- CW Angle Limit: the minimum value of Goal Position(Address 30, 31)
- CCW Angle Limit: the maximum value of Goal Position(Address 30, 31)

The following two modes can be set pursuant to the value of CW and CCW.

Operation Type	CW / CCW
Wheel Mode	both are 0
Joint Mode	neither at 0

The wheel mode can be used to wheel-type operation robots since motors of the robots spin infinitely.

The joint mode can be used to multi-joints robot since the robots can be controlled with specific angles.

The Highest Limit Temperature

Caution : Do not set the temperature lower/higher than the default value.

When the temperature alarm shutdown occurs, wait 20 minutes to cool the temperature before reuse.

Using the product when the temperature is high may and can cause damage.

The Lowest (Highest) Limit Voltage

It is the operation range of voltage.

50 to 250 (0x32 ~ 0x96) can be used. The unit is 0.1V.

For example, if the value is 80, it is 8V.

If Present Voltage (Address42) is out of the range, Voltage Range Error Bit (Bit0) of Status Packet is returned as '1' and Alarm is triggered as set in the addresses 17 and 18.

Max Torque

It is the torque value of maximum output. 0 to 1023 (0x3FF) can be used, and the unit is about 0.1%.

For example, Data 1023 (0x3FF) means that Dynamixel will use 100% of the maximum torque it can produce while Data 512 (0x200) means that Dynamixel will

use 50% of the maximum torque. When the power is turned on, Torque Limit (Addresses 34 and 35) uses the value as the initial value.

Status Return Level

It decides how to return Status Packet. There are three ways like the below table.

Value	Return of Status Packet
0	No return against all commands (Except PING Command)
1	Return only for the READ command
2	Return for all commands

When Instruction Packet is Broadcast ID, Status Packet is not returned regardless of Status Return Level.

Alarm LED

Alarm Shutdown

Dynamixel can protect itself by detecting errors occur during the operation. The errors can be set are as the table below.

Bit	Name	Contents

It is possible to make duplicate set since the function of each bit is run by the logic of 'OR'. That is, if 0X05 (binary 00000101) is set, both Input Voltage Error and Overheating Error can be detected.

If errors occur, in case of Alarm LED, the LED blinks; in case of Alarm Shutdown, the motor output becomes 0 % by making the value of Torque Limit(Address 34, 35) as 0.

RAM Area

Torque Enable

Value	Meaning
0	Keeps Torque from generating by interrupting the power of motor.
1	Generates Torque by impressing the power to the motor.

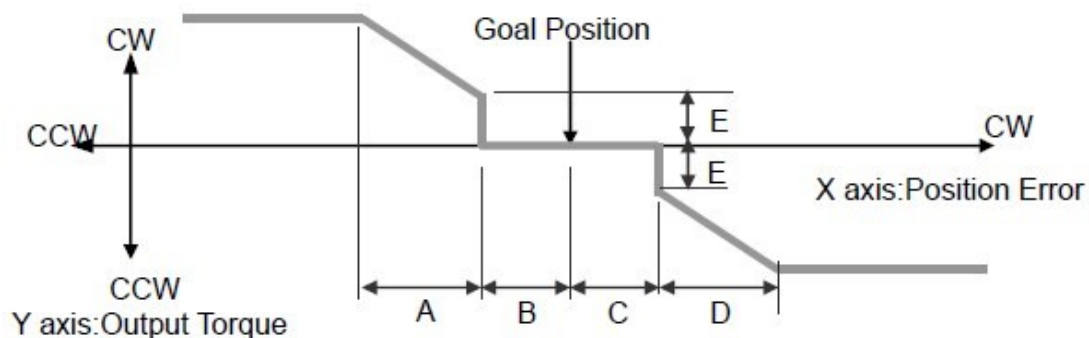
LED

Bit	Meaning
0	Turn OFF the LED
1	Turn ON the LED

Compliance

Compliance is to set the control flexibility of the motor.

The following diagram shows the relationship between output torque and position of the motor.



Compliance Margin

It exists in each direction of CW/CCW and means the error between goal position and present position.

The range of the value is 0~255, and the unit is the same as Goal Position.(Address 30,31) The greater the value, the more difference occurs.

Compliance Slope

It exists in each direction of CW/CCW and sets the level of Torque near the goal position.

Compliance Slope is set in 7 steps, the higher the value, the more flexibility is obtained.

Data representative value is actually used value. That is, even if the value is set to 25, 16 is used internally as the representative value.

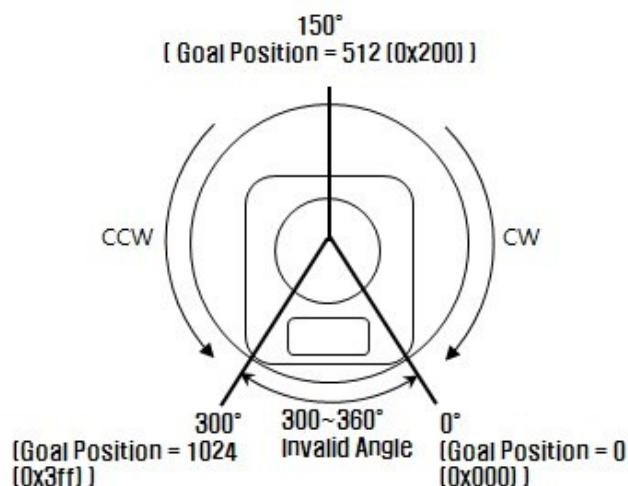
Step	Data Value	Data Representative Value
1	0 (0x00) ~ 3(0x03)	2 (0x02)
2	4(0x04) ~ 7(0x07)	4 (0x04)
3	8(0x08)~15(0x0 F)	8 (0x08)
4	16(0x10)~31(0x1F)	16 (0x10)
5	32(0x20)~63(0x3F)	32 (0x20)
6	64(0x40)~127(0x7F)	64 (0x40)
7	128(0x80)~254(0 xFE)	128 (0x80)

Goal Position

It is a position value of destination.

0 to 1023 (0x3FF) is available. The unit is 0.29 degree.

If Goal Position is out of the range, Angle Limit Error Bit (Bit1) of Status Packet is returned as '1' and Alarm is triggered as set in Alarm LED/Shutdown.



<The picture above is based on
the front of relevant model>

If it is set to Wheel Mode, this value is not used.

Moving Speed

It is a moving speed to Goal Position.

The range and the unit of the value may vary depending on the operation mode.

. Join Mode

0~1023 (0X3FF) can be used, and the unit is about 0.111rpm.

If it is set to 0, it means the maximum rpm of the motor is used without controlling the speed.

If it is 1023, it is about 114rpm.

For example, if it is set to 300, it is about 33.3 rpm.

Notes: Please check the maximum rpm of relevant model in Joint Mode.
Even if the motor is set to more than maximum rpm, it cannot generate the torque more than the maximum rpm.

. Wheel Mode

0~2047(0X7FF) can be used, the unit is about 0.1%.

If a value in the range of 0~1023 is used, it is stopped by setting to 0 while rotating to CCW direction.

If a value in the range of 1024~2047 is used, it is stopped by setting to 1024 while rotating to CW direction.

That is, the 10th bit becomes the direction bit to control the direction.

In Wheel Mode, only the output control is possible, not speed.

For example, if it is set to 512, it means the output is controlled by 50% of the maximum output.

Torque Limit

It is the value of the maximum torque limit.

0 to 1023 (0x3FF) is available, and the unit is about 0.1%.

For example, if the value is 512, it is about 50%; that means only 50% of the maximum torque will be used.

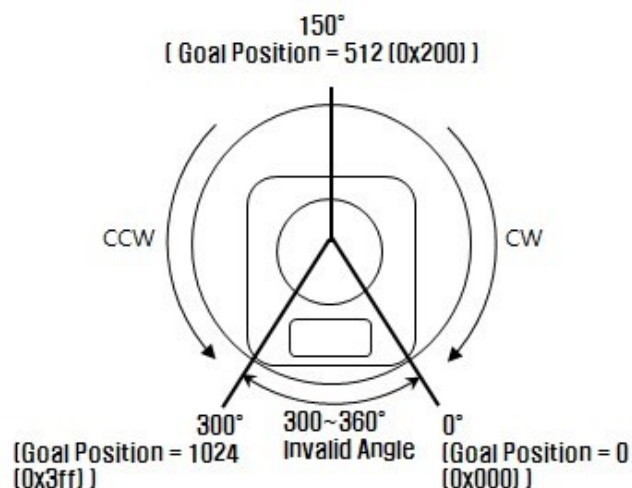
If the power is turned on, the value of Max Torque (Address 14, 15) is used as the initial value.

Notes: If the function of Alarm Shutdown is triggered, the motor loses its torque because the value becomes 0. At this moment, if the value is changed to the value other than 0, the motor can be used again.

Present Position

It is the current position value of Dynamixel.

The range of the value is 0~1023 (0x3FF), and the unit is 0.29 degree.



<The picture above is based on
the front of relevant model>

Caution: If it is set to Wheel Mode, the value cannot be used to measure the moving distance and the rotation frequency.

Present Speed

It is the current moving speed.

0~2047 (0X7FF) can be used.

If a value is in the range of 0~1023, it means that the motor rotates to the CCW direction.

If a value is in the range of 1024~2047, it means that the motor rotates to the CW direction.

That is, the 10th bit becomes the direction bit to control the direction, and 0 and 1024 are equal.

The unit of this value varies depending on operation mode.

. Joint Mode

The unit is about 0.111rpm.

For example, if it is set to 300, it means that the motor is moving to the CCW direction at a rate of about 33.3rpm. . Wheel Mode

The unit is about 0.1%.

For example, if it is set to 512, it means that the torque is controlled by 50% of the maximum torque to the CCW direction.

Present Load

It means currently applied load.

The range of the value is 0~2047, and the unit is about 0.1%.

If the value is 0~1023, it means the load works to the CCW direction.

If the value is 1024~2047, it means the load works to the CW direction.

That is, the 10th bit becomes the direction bit to control the direction, and 1024 is equal to 0.

For example, the value is 512, it means the load is detected in the direction of CCW about 50% of the maximum torque.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction	Data (Load Ratio)									

Load Direction = 0 : CCW Load, Load Direction = 1: CW Load

Notes: Present load is inferred from the output value, not from torque sensor, etc.

For this reason, it cannot be used to measure weight or torque but to detect which direction the force works.

Present Voltage

It is the size of the current voltage supplied.

This value is 10 times larger than the actual voltage. For example, when 10V is supplied, the data value is 100 (0x64)

Present Temperature

It is the internal temperature of Dynamixel in Celsius.

Data value is identical to the actual temperature in Celsius. For example, if the data

value is 85 (0x55), the current internal temperature is 85°C.

Registered Instruction

Value	Meaning
0	There are no commands transmitted by REG WRITE
1	There are commands transmitted by REG WRITE.

Notes: If ACTION command is executed, the value is changed into 0.

Moving

Value	Meaning
0	Goal position command execution is completed.
1	Goal position command execution is in progress.

Lock

Value	Meaning
0	EEPROM area can be modified.
1	EEPROM area cannot be modified.

Caution: If Lock is set to 1, the power must be turned off and then turned on again to change into 0.

Punch

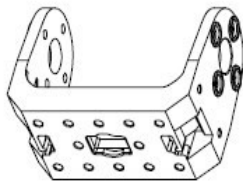
Current to drive motor is at minimum.

Can choose vales from 0x20 to 0x3FF.

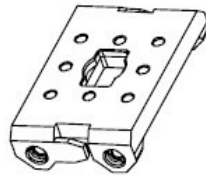
Option Frame

Basic-Offered Frames

Basic-offered frames with AX-12A are as follows.



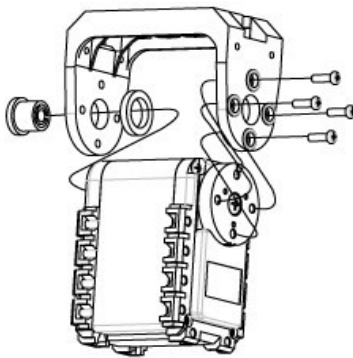
FP04-F2



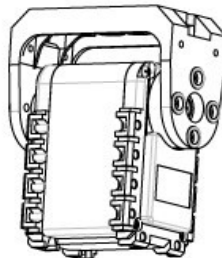
FP04-F3

Applying F2

F2 is applied as below.



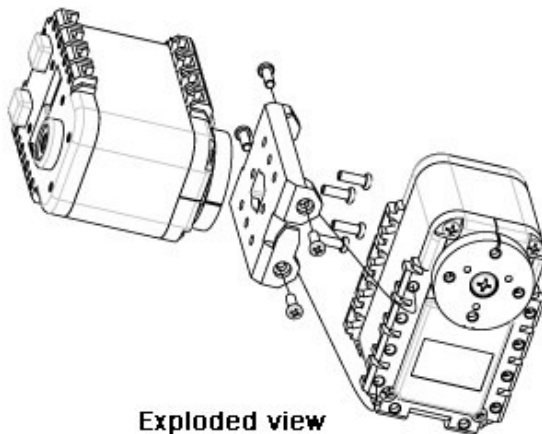
Exploded view



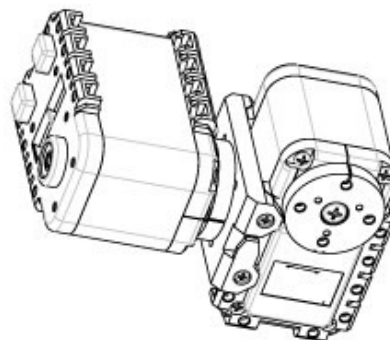
Assembled

Applying F3

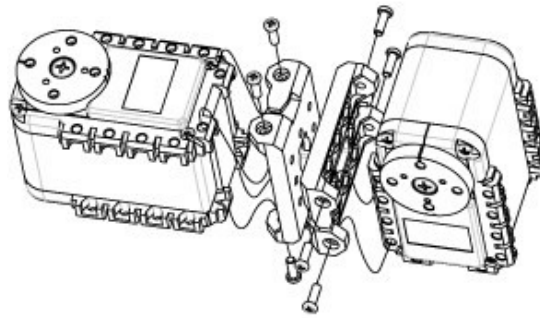
F3 is applied as below. F3 can be connected to 3 sides of AX-12A: Left, Right, and the bottom.



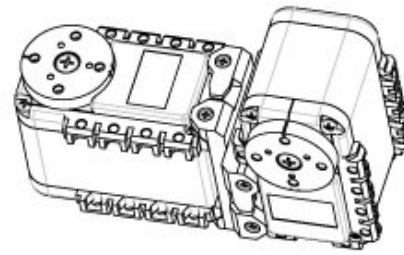
Exploded view



Assembled

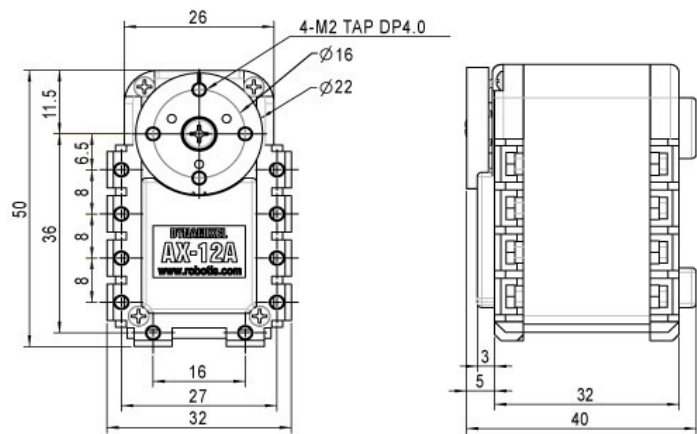


Exploded view



Assembled

Dimension



ANEXO C – Plataforma comercial tomada como base para o modelo simulado. 6- Axis Positioner with Controller



COST- EFFICIENT HEXAPOD

- + Six degrees of freedom, travel ranges to 100 mm / 60°
- + Load capacity to 20 kg
- + Velocity under full load to 20 mm/ s
- + Repeatability up to $\pm 1 \mu\text{m}$
- + MTBF 20,000 h
- + Works in any orientation
- + Rapid response behavior
- + Sophisticated controller using vector algorithms, virtual pivot point
- + Comprehensive software package

Standard- class 6- axis positioning system

Parallel- kinematic design for six degrees of freedom making it significantly more compact and stiff than serial- kinematic systems, higher dynamic range, no moved cables: Higher reliability, reduced friction

Direct drive with brushless DC motors (BLDC) Indirect measuring principle

Rotary encoder on motor shaft

Powerful digital controller, open software architecture

User- defined, stable pivot point, software- selectable. Positions commanded in Cartesian coordinates. Macro programming. Open source LabVIEW driver set. Work space simulation software. Optional interface for PLC control

Fields of application

Industry and research. For life science, biotechnology, automation, micromachining

Specifications

	H-820.D2	Unit	Tolerance
Active axes	X, Y, Z, θX , θY , θZ		
Motion and positioning			
Travel range* X, Y	± 50	mm	
Travel range* Z	± 25	mm	
Travel range* θX , θY	± 15	°	
Travel range* θZ	± 30	°	
Actuator drive	Torque motor, brushless (BLDC)		
Single- actuator design resolution	0.2	μm	typ.
Min. incremental motion X, Y, Z	10	μm	typ.
Min. incremental motion θX , θY , θZ	25	μrad	typ.
Repeatability X, Y	± 2	μm	typ.
Repeatability Z	± 1	μm	typ.
Repeatability θX , θY	± 15	μrad	typ.
Repeatability θZ	± 30	μrad	typ.
Backlash X, Y	30	μm	typ.
Backlash Z	10	μm	typ.
Backlash θX , θY	100	μrad	typ.
Backlash θZ	300	μrad	typ.
Max. velocity X, Y, Z	20	mm/ s	
Max. velocity θX , θY , θZ	200	mrad/ s	
Mechanical properties			
Load (base plate horizontal)	20	kg	max.
Load (base plate in any orientation)	10	kg	max.
Holding force (base plate horizontal)	200	N	max.
Holding force (base plate in any orientation)	100	N	max.
Miscellaneous			
Operating temperature range	0 to +50	°C	
Material	Aluminum		
Mass	15	kg	$\pm 5\%$
Cable length	3	m	$\pm 10\text{ mm}$
Recommended controller	C-887		
Operating voltage	100 to 240 VAC, 50/60 Hz		

Technical data specified at $20 \pm 3\text{ }^{\circ}\text{C}$.

Ask about custom designs!

* The travel ranges of the individual coordinates (X, Y, Z, θX , θY , θZ) are interdependent. The data for each axis in this table shows its maximum travel, where all other axes are at their zero positions. If the other linear or rotational coordinates are not zero, the available travel may be less.

Order Information

H-820.D2

Hexapod Microrobot, Basic Version, 20 mm/ s, 20 kg Load, Sub- D

Connector, Cable Set 3 m Ask about custom designs!

Recommended controller

C-887.52

6D Hexapod Controller, TCP/ IP, RS-232, Bench- Top Device, Control of Two Additional Servo- Motor Axes Included

Hexapod system with fieldbus interface

H-820.D31

Hexapod Microrobot, Basic Version, 20 mm/ s, 20 kg Load, Cable Set 3 m, with 6- D Hexapod Controller, EtherCAT® Interface

Controllers / Drivers / Amplifiers

C-887 Controller for Hexapod Positioning Systems

Related Products

H-824 6- Axis Hexapod

H-840 6- Axis Hexapod

H-850 6- Axis Hexapod

H-860KMAG High- Dynamics Hexapod

Technology

Hexapods – Parallel- Kinematics Positioning Systems | Hexapod platforms are used for precision positioning and alignment of loads in all six degrees of freedom, three linear axes, and three rotational axes. Learn more ...

Hexapod as Motion Simulator | Motion simulators have higher motion dynamics requirements (shakers). Learn more ...

Drawings / Images

H-820.D2, dimensions in mm

